XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos
24 a 28 de maio de 2010
Gramado, RS

VIII Workshop em Clouds, Grids e Aplicações (WCGA)

Editora
Sociedade Brasileira de Computação (SBC)

Organizadores
Bruno Schulze (LNCC)
Alfredo Goldman (USP)
Antônio Jorge Gomes Abelém (UFPA)
Luciano Paschoal Gaspary (UFRGS)
Marinho Pilla Barcellos (UFRGS)

Realização
Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)

Promoção
Sociedade Brasileira de Computação (SBC)
Laboratório Nacional de Redes de Computadores (LARC)

Capa: Josué Klafke Sperb
Produção Editorial: Flávio Roberto Santos, Roben Castagna Lunardi, Matheus Lehmann, Rafael Santos Bezerra, Luciano Paschoal Gaspary e Marinho Pilla Barcellos.

Cópias Adicionais:
Sociedade Brasileira de Computação (SBC)
Av. Bento Gonçalves, 9500 - Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia - CEP 91.509-900 - Porto Alegre - RS
Fone: (51) 3308-6835
E-mail: sbc@sbc.org.br

Dados Internacionais de Catalogação na Publicação (CIP)

Workshop em Clouds, Grids e Aplicações (8. : 2010 : Gramado, RS).

Anais / VIII Workshop em Clouds, Grids e Aplicações ; organizadores Bruno Schulze... et al. − Porto Alegre : SBC, c2010.

157 p.

ISSN 2177-496X

1. Redes de computadores. 2. Sistemas distribuídos. I. Schulze, Bruno. II. Título.

Promoção

Sociedade Brasileira de Computação (SBC)

Diretoria

Presidente
José Carlos Maldonado (USP)

Vice-Presidente
Marcelo Walter (UFRGS)

Diretor Administrativo
Luciano Paschoal Gaspary (UFRGS)

Diretor de Finanças
Paulo Cesar Masiero (USP)

Diretor de Eventos e Comissões Especiais
Lisandro Zambenedetti Granville (UFRGS)

Diretora de Educação
Mirella Moura Moro (UFMG)

Diretora de Publicações
Karin Breitman (PUC-Rio)

Diretora de Planejamento e Programas Especiais
Ana Carolina Salgado (UFPE)

Diretora de Secretarias Regionais
Thais Vasconcelos Batista (UFRN)

Diretor de Divulgação e Marketing
Altigran Soares da Silva (UFAM)

Diretor de Regulamentação da Profissão
Ricardo de Oliveira Anido (UNICAMP)

Diretor de Eventos Especiais
Carlos Eduardo Ferreira (USP)

Diretor de Cooperação com Sociedades Científicas
Marcelo Walter (UFRGS)

Promoção

Conselho

Mandato 2009-2013
Virgílio Almeida (UFMG)
Flávio Rech Wagner (UFRGS)
Silvio Romero de Lemos Meira (UFPE)
Itana Maria de Souza Gimenes (UEM)
Jacques Wainer (UNICAMP)

Mandato 2007-2011
Cláudia Maria Bauzer Medeiros (UNICAMP)
Roberto da Silva Bigonha (UFMG)
Cláudio Leonardo Lucchesi (UNICAMP)
Daltro José Nunes (UFRGS)
André Ponce de Leon F. de Carvalho (USP)

Suplentes - Mandato 2009-2011
Geraldo B. Xexeo (UFRJ)
Taisy Silva Weber (UFRGS)
Marta Lima de Queiroz Mattoso (UFRJ)
Raul Sidnei Wazlawick (UFSC)
Renata Vieira (PUCRS)

Laboratório Nacional de Redes de Computadores (LARC)

Diretoria

Diretor do Conselho Técnico-Científico
Artur Ziviani (LNCC)

Diretor Executivo
Célio Vinicius Neves de Albuquerque (UFF)

Vice-Diretora do Conselho Técnico-Científico
Flávia Coimbra Delicato (UFRN)

Vice-Diretor Executivo
Luciano Paschoal Gaspary (UFRGS)

Membros Institucionais
CEFET-CE, CEFET-PR, IME, INPE/MCT, LNCC, PUCPR, PUC-RIO, SESU/MEC, UECE, UERJ, UFAM, UFBA, UFC, UFCG, UFES, UFF, UFMG, UFPA, UFPB, UFPE, UFPR, UFRGS, UFRJ, UFRN, UFSC, UFSCAR, UNICAMP, UNIFACS, USP.

Realização

Comitê de Organização

Coordenação Geral
Luciano Paschoal Gaspary (UFRGS)
Marinho Pilla Barcellos (UFRGS)

Coordenação do Comitê de Programa
Luci Pirmez (UFRJ)
Thaís Vasconcelos Batista (UFRN)

Coordenação de Palestras e Tutoriais
Lisandro Zambenedetti Granville (UFRGS)

Coordenação de Painéis e Debates
José Marcos Silva Nogueira (UFMG)

Coordenação de Minicursos
Carlos Alberto Kamienski (UFABC)

Coordenação de Workshops
Antônio Jorge Gomes Abelém (UFPA)

Coordenação do Salão de Ferramentas
Nazareno Andrade (UFCG)

Comitê Consultivo
Artur Ziviani (LNCC)
Carlos André Guimarães Ferraz (UFPE)
Célio Vinicius Neves de Albuquerque (UFF)
Francisco Vilar Brasileiro (UFCG)
Lisandro Zambenedetti Granville (UFRGS)
Luís Henrique Maciel Kosmalski Costa (UFRJ)
Marcelo Gonçalves Rubinstein (UERJ)
Nelson Luis Saldanha da Fonseca (UNICAMP)
Paulo André da Silva Gonçalves (UFPE)

Realização

Organização Local
Adler Hoff Schmidt (UFRGS)
Alan Mezzomo (UFRGS)
Alessandro Huber dos Santos (UFRGS)
Bruno Lopes Dalmazo (UFRGS)
Carlos Alberto da Silveira Junior (UFRGS)
Carlos Raniery Paula dos Santos (UFRGS)
Cristiano Bonato Both (UFRGS)
Flávio Roberto Santos (UFRGS)
Jair Santanna (UFRGS)
Jéferson Campos Nobre (UFRGS)
Juliano Wickboldt (UFRGS)
Leonardo Richter Bays (UFRGS)
Lourdes Tassinari (UFRGS)
Luís Armando Bianchin (UFRGS)
Luis Otávio Luz Soares (UFRGS)
Marcos Ennes Barreto (UFRGS)
Matheus Brenner Lehmann (UFRGS)
Pedro Arthur Pinheiro Rosa Duarte (UFRGS)
Pietro Biasuz (UFRGS)
Rafael Pereira Esteves (UFRGS)
Rafael Kunst (UFRGS)
Rafael Santos Bezerra (UFRGS)
Ricardo Luis dos Santos (UFRGS)
Roben Castagna Lunardi (UFRGS)
Rodolfo Stoffel Antunes (UFRGS)
Rodrigo Mansilha (UFRGS)
Weverton Luis da Costa Cordeiro (UFRGS)

Mensagem dos Coordenadores Gerais

Bem-vindo(a) ao XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2010)! Esta edição do simpósio está sendo realizada de 24 a 28 de maio de 2010 na pitoresca cidade de Gramado, RS. Promovido pela Sociedade Brasileira de Computação (SBC) e pelo Laboratório Nacional de Redes de Computadores (LARC) desde 1983, o SBRC 2010 almeja não menos que honrar com uma tradição de quase 30 anos: ser reconhecido como o mais importante evento científico em redes de computadores e sistemas distribuídos do país, e um dos mais concorridos em Informática. Mais do que isso, pretende estimular intercâmbio de idéias e discussões qualificadas, aproximá-lo(a) de temas de pesquisa efervescentes e fomentar saudável aproximação entre estudantes, pesquisadores, professores e profissionais.

Para atingir os objetivos supracitados, reunimos um grupo muito especial de professores atuantes em nossa comunidade que, com o nosso apoio, executou com êxito a tarefa de construir um **Programa Técnico** de altíssima qualidade. O SBRC 2010 abrange as seguintes atividades: 20 sessões técnicas de artigos completos, cobrindo uma grande gama de problemas em redes de computadores e sistemas distribuídos; 2 sessões técnicas para apresentações de ferramentas; 5 minicursos ministrados de forma didática, por professores da área, sobre temas atuais; 3 palestras e 3 tutoriais sobre tópicos de pesquisa avançados, apresentados por especialistas nacionais e estrangeiros; e 3 painéis versando sobre assuntos de relevância no momento. Completa a programação técnica a realização de 8 *workshops* satélites em temas específicos: WRNP, WGRS, WTR, WSE, WTF, WCGA, WP2P e WPEIF. Não podemos deixar de ressaltar o **Programa Social**, organizado em torno da temática "*vinho*", simbolizando uma comunidade de pesquisa madura e que, com o passar dos anos, se aprimora e refina cada vez mais.

Além da ênfase na qualidade do programa técnico e social, o SBRC 2010 ambiciona deixar, como marca registrada, seu esforço na busca por excelência organizacional. Tal tem sido perseguido há mais de dois anos e exigido muita determinação, dedicação e esforço de uma equipe afinada de organização local, composta por estudantes, técnicos administrativos e professores. O efeito desse esforço pode ser percebido em elementos simples, mas diferenciais, tais como uniformização de datas de submissão de trabalhos, portal *sempre* atualizado com as últimas informações, comunicação sistemática com potenciais participantes e pronto atendimento a qualquer dúvida. O nosso principal objetivo com essa iniciativa foi e continua sendo oferecer uma elevada *qualidade de experiência* a você, colega participante!

Gostaríamos de agradecer aos membros do Comitê de Organização Geral e Local que, por conta de seu trabalho voluntário e incansável, ajudaram a construir um evento que julgamos de ótimo nível. Gostaríamos de agradecer, também, à SBC, pelo apoio prestado ao longo das muitas etapas da organização, e aos patrocinadores, pelo incentivo à divulgação de atividades de pesquisa conduzidas no País e pela confiança depositada neste fórum. Por fim, nossos agradecimentos ao Instituto de Informática da UFRGS, por viabilizar a realização, pela quarta vez, de um evento do porte do SBRC.

Sejam bem-vindos à Serra Gaúcha para o "SBRC do Vinho"! Desejamos que desfrutem de uma semana agradável e proveitosa!

Luciano Paschoal Gaspary
Marinho Pilla Barcellos
Coordenadores Gerais do SBRC 2010

Mensagem dos Coordenadores do WCGA

O VIII Workshop em Clouds, Grids e Aplicações (WCGA 10) em conjunto com o 28o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2010), em Gramado, RS, tem por objetivo atuar como um fórum para apresentações técnicas de pesquisas em andamento e atividades relevantes nas áreas de infra-estrutura e infra-estrutura virtualizada, serviços e aplicações em áreas diversas, congregando pesquisadores e profissionais que atuam ativamente nessas áreas. O workshop também procura estabelecer redes colaborativas multi-institucionais e grupos de competência técnico-científica, bem como fortalecer atividades em andamento.

As primeiras edições do WCGA (2003 - 2004 - 2005), foram realizadas no LNCC, em Petrópolis-RJ. As edições de 2006, 2007, 2008 e 2009 foram realizadas em conjunto com o Simpósio Brasileiro de Redes de Computadores - SBRC em Curitiba (PR), Belém (PA) e Rio de Janeiro (RJ) e Refice (PE). Estas edições anteriores do WCGA receberam uma combinação interessante de diferentes áreas de computação em grids como infra-estrutura, middleware, redes e aplicações, com apresentações técnicas e posters fomentando discussões em vários tópicos interessantes, incluindo grids clássicos, grids baseados em objetos e serviços, escalonamento, e aplicações, tais como bioinformática; meteorologia e governo eletrônico, entre outros. Estas edições geraram um interesse crescente na comunidade e nos baseamos nesta tradição para esta edição de 2010.

A oitava edição do WCGA teve dezesseis (16) trabalhos submetidos sendo onze (11) aceitos como artigos completos. Os autores apontaram problemas e soluções em um ou mais temas dentre aqueles identificados para o workshop.

Finalmente, a coordenação do workshop gostaria de agradecer a todos os autores e participantes pela apresentação dos trabalhos e contribuições de pesquisa em clouds, grids e aplicações, agradecer a todos os membros dos comitês de programa e de organização pela ajuda na elaboração do programa do workshop e nas atividades, e adicionalmente agradecer os apoios de LNCC, USP, UFRGS e SBC.

Bruno Schulze
Alfredo Goldman
Coordenadores do WCGA 2010

Comitê de Programa do WCGA

Antônio Roberto Mury, Laboratório Nacional de Computação Científica (LNCC)
Antônio Tadeu A Gomes, Laboratório Nacional de Computação Científica (LNCC)
Artur Ziviani, Laboratório Nacional de Computação Científica (LNCC)
Celso Mendes, University of Illinois at Urbana-Champaign
Cesar De Rose, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Claudio Geyer, Universidade Federal do Rio Grande do Sul (UFRGS)
Cristina Boeres, Universidade Federal Fluminense (UFF)
Edmundo Madeira, Universidade Estadual de Campinas (UNICAMP)
Fabíola Greve, Universidade Federal da Bahia (UFBA)
Fabio Costa, Universidade Federal de Goiás (UFG)
Fabio Porto, Laboratório Nacional de Computação Científica (LNCC)
Fabricio Alves Barbosa da Silva, TU Lisboa
Francisco José da Silva e Silva, Universidade Federal do Maranhão (UFMA)
Hélio Guardia, Universidade Federal de São Carlos (UFSCar)
Jose Neuma de Souza, Universidade Federal do Ceará (UFC)
Leonel Sousa, TU Lisboa
Lucia Drummond, Universidade Federal Fluminense (UFF)
Luis Carlos De Bona, Universidade Federal do Paraná (UFPR)
Marcelo Pasin, TU Lisboa
Philippe Navaux, Universidade Federal do Rio Grande do Sul (UFRGS)
Rapahel Camargo, Universidade Federal do ABC (UFABC)
Rossana M de Castro Andrade, Universidade Federal do Ceará (UFC)
Vinod Rebello, Universidade Federal Fluminense (UFF)
Wagner Meira Jr., Universidade Federal de Minas Gerais (UFMG)

Revisores do WCGA

Antônio Roberto Mury, Laboratório Nacional de Computação Científica (LNCC)
Antônio Tadeu A Gomes, Laboratório Nacional de Computação Científica (LNCC)
Artur Ziviani, Laboratório Nacional de Computação Científica (LNCC)
Carlos Senna, Universidade Estadual de Campinas (UNICAMP)
Celso Mendes, University of Illinois at Urbana-Champaign
Cesar De Rose, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Claudio Geyer, Universidade Federal do Rio Grande do Sul (UFRGS)
Cristina Boeres, Universidade Federal Fluminense (UFF)
Edmundo Madeira, Universidade Estadual de Campinas (UNICAMP)
Fabíola Greve, Universidade Federal da Bahia (UFBA)
Fabio Costa, Universidade Federal de Goiás (UFG)
Fabio Porto, Laboratório Nacional de Computação Científica (LNCC)
Fabricio Alves Barbosa da Silva, TU Lisboa
Francisco José da Silva e Silva, Universidade Federal do Maranhão (UFMA)
Guna Santos, Universidade Federal da Bahia (UFBA)
Hélio Guardia, Universidade Federal de São Carlos (UFSCar)
Jose Neuma de Souza, Universidade Federal do Ceará (UFC)
Leonel Sousa, TU Lisboa
Lucia Drummond, Universidade Federal Fluminense (UFF)
Luis Carlos De Bona, Universidade Federal do Paraná (UFPR)
Luiz Fernando Bittencourt, Universidade Estadual de Campinas (UNICAMP)
Marcelo Pasin, TU Lisboa
Marco Netto, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Nicolas Maillard, Universidade Federal do Rio Grande do Sul (UFRGS)
Philippe Navaux, Universidade Federal do Rio Grande do Sul (UFRGS)
Rapahel Camargo, Universidade Federal do ABC (UFABC)
Rodrigo Calheiros, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Rossana M de Castro Andrade, Universidade Federal do Ceará (UFC)
Tiago Ferreto, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Vinod Rebello, Universidade Federal Fluminense (UFF)
Wagner Meira Jr., Universidade Federal de Minas Gerais (UFMG)

Sumário

VIII Workshop em Clouds, Grids e Aplicações

♦

Sessão Técnica 1

Infraestrutura Virtualizada

A Survey on Open-source Cloud Computing Solutions

Patrícia Takako Endo[1], Glauco Estácio Gonçalves[1], Judith Kelner[1], Djamel Sadok[1]

[1]Universidade Federal de Pernambuco – UFPE
Caixa Postal 7851 – 50732-970 – Recife – PE

{patricia,glauco,jk,jamel}@gprt.ufpe.br

Abstract. *Cloud computing is an attractive computing model since it allows for resources to be provisioned according on a demand basis, i.e., cloud users can rent resources as they become necessary. This model motivated several academic and non-academic institutions to develop open-source cloud solutions. This paper presents and discusses the state-of-the of open-source solutions for cloud computing. The authors hope that the observation and classification of such solutions can leverage the cloud computing research area providing a good starting point to cope with some of the problems present in cloud computing environments.*

1. Introduction

At present, it is common to access content across the Internet with little reference to the underlying hosting infrastructure, which is formed by data centers maintained by content providers. The entire technology used to provide such level of locality transparency offers also a new model for the provision of computing services, stated as Cloud Computing. In such a model, enterprises put their applications in the cloud – a very large computational entity from a developer´s perspective – without concern for where the services are actually hosted or how they are delivered. Through access to slice of computational power over a scalable network of nodes, enterprises can reduce or eliminate costs associated with internal infrastructure for the provision of their services.

A further viewpoint of cloud users, in addition to costs reduction, comes from the fact that the cloud computing model can be attractive since it allows resources to be provisioned according to the enterprise needs. This is in contrast, to traditional practices, where resources are often dimensioned according to the worst case use and peak scenarios. Hence cloud users can rent resources as they become necessary, in a much more scalable and elastic way. Moreover, enterprises can transfer operational risks to cloud providers. In the viewpoint of cloud providers, the model offers a way for better utilization of their own infrastructure. Authors in [Armbrust et al 2009] point that this model benefits from a form of statistical multiplexing, since it allocates resources for several users concurrently. This statistical multiplexing of data centers is guaranteed through several decades of research in many areas as: distributed computing, grid computing, web technologies, service computing, and virtualization. Several authors ([Armbrust et al 2009], [Vaquero et al 2008], [Buyya et al 2009]) agree that from such research areas virtualization brought the key technologies to leverage cloud computing.

Despite the increasingly widespread use of the Cloud Computing term, there is no formal definition for it yet. In a recent paper [Vaquero et al 2008], authors review the

cloud literature for a minimum set of characteristics that cloud solutions must present. But they were not able to find even a single common feature in literature definitions. They note that the set of characteristics that at most are similar to a minimum common definition are: scalability, pay-per use model, and virtualization. Finally, using these features the authors gave their own definition: “Clouds are a large pool of easily usable and accessible virtualized resources. These resources can be dynamically reconfigured to adjust to a variable load, allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLA”.

Since the popularization of the cloud computing term in 2007, with IBM Blue Cloud [Vouk 2008], several enterprises become cloud computing providers: Amazon and their Elastic Compute Cloud (EC2)[1], Google with Google App Engine[2], Microsoft with their Windows Azure Platform[3], Salesforce and their Force.com[4] and so on. Though these solutions fit the cloud computing definition they differ in their programmability. A concept borrowed from the network virtualization area [Chowdhury and Boutaba 2010], programmability is relative to the programming features a network element offers to developers, measuring how much freedom the developer has to manipulate resources and/or devices.

This concept can be applied to compare cloud computing solutions. More programmable clouds offer environments where developers are free to choose their own programming paradigm, languages, and platforms, having total control over their virtualized resources. Less programmable clouds restrict developers in some way: forcing a set of programming languages, or allowing support for only one application paradigm. On the other hand, a high level of programmability turns hard the cloud management because the cloud provider will have a much more heterogeneous environment to manage. For example, if a cloud provider allows their users to use any operating system in a virtual machine, the cloud operator will have to cope with a large number of solutions to provide fault-tolerance services. Moreover, less programmable solutions abstract some operational issues (processes communication, network access, storage functions, etc.) through some type of middleware. An instance of a cloud solution with high programmability is the Amazon EC2, where users can choose their operating system from a set of supported operating systems given by Amazon and they can configure their virtual machines to work as they see fit. The Google App Engine is an example of a less programmable solution, since it restricts developers to the Web paradigm and to some programming languages.

A common characteristic present in all the cited cloud solutions is that cloud owners understandably avoid revealing the underlying features of their solution, since this is seen as part of their strategic information. Despite this, one may point out the considerable efforts made by several academic and non-academic institutions to develop open-source cloud solutions. The authors hope that the observation and classification of

[1] http://aws.amazon.com/ec2/

[2] http://code.google.com/intl/appengine/appengine/

[3] http://www.microsoft.com/windowsazure/

[4] http://www.salesforce.com/platform/

such solutions can leverage further the cloud computing research area providing a good starting point to discovering different approaches to the problems present in cloud computing environments.

This paper presents and discusses the state-of-the of open-source solutions for cloud computing. The remainder of this paper is organized as follows. Next section introduces the main challenges developers face to create cloud computing solutions. Section III shows efforts in standardizing cloud computing interfaces for interoperability. Section IV introduces the main open-source cloud architectures and Section V makes comparisons between the architectures. Finally, a conclusion of this paper and future works are presented in Sections VI.

2. Challenges

The development of cloud computing solutions brings several technical challenges to cloud developers. These challenges can be grouped in three main areas: negotiation, decision, and operation. In the negotiation area, these are the challenges relative to how application developers interface with the cloud as well as the description of the cloud offerings. It includes also the definition of the programmability level that the cloud solution will offer. The decision area copes with the main problem that clouds faces behind the scenes: How virtual resources can be scheduled to meet user requirements, for example? Last, the operation area is associated with the enforcement of decisions and the communication between cloud elements. The following sub-sections discuss in details the main challenges in each one of these areas.

2.1. Negotiation

The negotiation area concerns itself with challenges relative to the interface between the application developers and the cloud. Generally, the interface between the cloud and application developers assumes the form of an Application Programming Interface (API), but, depending on the programmability level offered by the cloud, this API can be implemented in several ways ranging from a web-service based toolkit to control virtual machines in the cloud to a set of programming primitives used to develop distributed applications in the cloud. In addition to these basic functions, such APIs must allow developers to request – and control, possibly – additional functionalities offered by a cloud operator like service quality assessment, load balance, elastic application growth, backup strategies, and so on. There are still some other requirements that can be considered in the cloud API like geographical restrictions enforced to allocate virtual machines due to legal issues. One may think of some type of content or application that is strategically limited to a country or a region for copyright or security reasons.

At the present, APIs and the negotiation process offered by cloud providers follow a semi-automatic scheme where a human interacts with the cloud through programming primitives or a visual interface. But, next generation clouds can offer sophisticated ways to interact with human users through high-level abstractions and service-level policies. Such an interface type will need some formalism to specify both cloud offerings and application requirements as well as offering the support for an automatic negotiation process.

2.2. Decision

The main target of any cloud operator is to schedule developer applications aiming for the maximum utilization of cloud resources. A developer´s application covers, beyond the actual code, some additional information about application´s needs and services negotiated previously. In other words, one can abstract this additional information to some type of network virtualization demand with a topology formed by virtual nodes where the application runs and virtual links for communication. Thus, the cloud operator problem turns into that of selecting the best suitable physical resources to accommodate these virtual resources.

This careful mapping requires advanced strategies. The first challenge imposed by this optimization problem is that it is NP-hard [Andersen 2002] and hence any useful solution would need to relax some of its problem conditions and constraints to obtain an approximate solution in a polynomial time. The second challenge is to meet all the clauses negotiated with the developer. Depending on the nature of such contract, application scheduling algorithms will cope with quality restrictions, jurisdiction restrictions, elastic growth, and so on.

2.3. Operation

Metaphorically, one can say that while in the decision area the cloud operator must identify solutions for the “brain” of the cloud, in the operation area it must attack the problems of the “limbs” of the cloud, i.e., they must provide some form to enforce decisions. The enforcement here covers the communication protocols and the configuration of cloud elements.

A communication protocol can be used to monitor and reserve resources in the cloud. The cloud is composed by different elements like processing servers, switches, routers, links and storage components. Due to such heterogeneity, the communication between the decision-maker and elements puts a challenge on cloud design. Overall, existing cloud solutions use Web Services to provide communication with processing and storage nodes, but many communication elements do not support such implementations. Thus, cloud architects are using traditional traffic engineering protocols to provide reservation of network elements. One possible idea to cope with this challenge is to use smart communication nodes with an open interface to create new services in the node the emerging Openflow-enabled switches [McKeown et al 2008].

Node communication is just one part of the problem; the other one is to configure this. Here, the recent advances in server virtualization have provided several solutions for operators to benefit from.

3. Standardization efforts

A considerable challenge present in many of the raised discussions around the cloud is related to the need for standardization. All the three areas presented in Section 2 face the standardization challenge in some way, but the main challenge occurs in the negotiation area. Currently, cloud providers offer proprietary interfaces to access their services. This locks users within a given provider as they cannot migrate their applications and services easily between cloud providers [Buyya et al 2009]. It is hoped that cloud

providers see such a problem and work together to offer a standardized API based on open standards like SOAP and REST.

An important effort in the standardization comes from the Open Cloud Manifesto [OpenCloud 2009]. This is an initiative supported by hundreds of companies that aims to discuss with cloud organizations a way to produce open standards for cloud computing. Their major doctrines are collaboration and coordination of efforts on the standardization, adoption of open standards wherever appropriate, and the development of standards based on customer requirements. Participants of the Open Cloud Manifesto through the Cloud Computing Use Case group produced an interesting white paper [OpenCloud 2010] highlighting the requirements that need to be standardized in a cloud environment to ensure interoperability in the most typical scenarios of interaction – Use Cases – in cloud computing.

4. Open-source solutions for Cloud Computing

Due to the large growth of cloud computing, there are several solutions in this area. This article is focused on open source solutions, highlighting their main characteristics and architectures proposed.

4.1. Xen Cloud Platform (XCP)

The Xen hypervisor [Citrix Systems 2010b] is a solution for infrastructure virtualization that provides an abstraction layer between servers' hardware and the operating system. A Xen hypervisor allows each physical server to run several "virtual servers" handling the operating system and its applications from the underlying physical server. The Xen solution is used by many cloud solutions such as Amazon EC2, Nimbus and Eucalyptus.

Recently, Xen.org announced the Xen Cloud Platform (XCP) [Citrix Systems 2010a] as a solution for cloud infrastructure virtualization. But, differently from existent open source cloud solutions, XCP does not provide the overall architecture for cloud services. Their goal is to provide a tool to cope with automatic configuration and maintenance of cloud platforms [Citrix Systems 2010c].

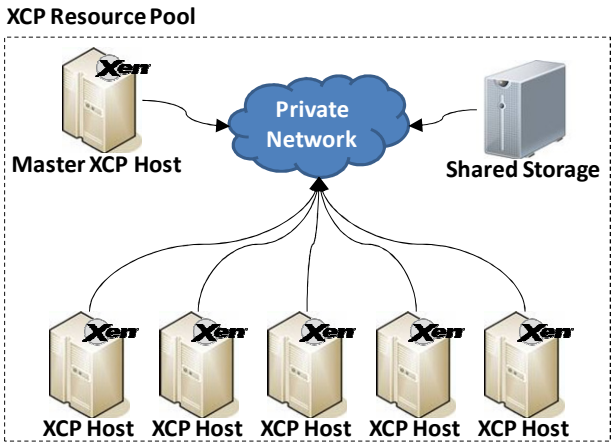


Figure 1. XCP Architecture

The XCP architecture (Figure 1) is based on the **XCP hosts** that are responsible to host the virtual machines. According to [Xen.org. 2009], these hosts are aggregated in

a **XCP resource pool** and using a **Shared Storage** the virtual machines can be started and restarted on any XCP host. The **Master XCP host** offers an administration interface and forwards command messages to others XCP hosts.

4.2. Nimbus

Nimbus [Keahey 2009] is an open source solution (licensed under the terms of the Apache License) to turn clusters into an Infrastructure as a Service (IaaS) for Cloud Computing focusing mainly on scientific applications.

This solution gives to users the possibility to allocate and configure remote resources by deploying VMs – known as Virtual Workspace Service (VWS). A VWS is a VM manager that different frontends can invoke.

To deploy applications, Nimbus offers a "cloudkit" configuration that consists of a manager service hosting and an image repository. The workspace components are shown in Figure 2.

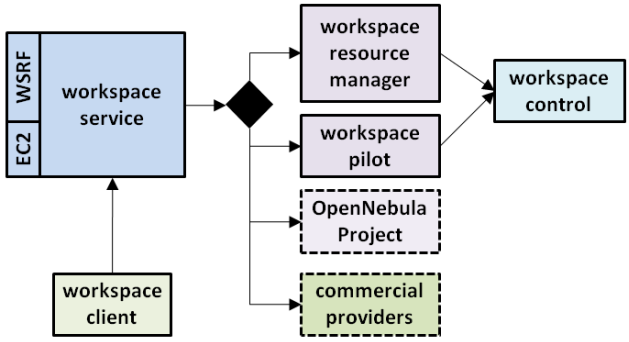


Figure 2. Nimbus workspace components [Keahey 2009]

- **Workspace service**: is web services based and provides security with the GSI authentication and authorization. Currently, Nimbus supports two frontends: Amazon EC2 and WSRF.
- **Workspace control**: is responsible for controlling VM instances, managing and reconstructing images, integrating a VM to the network and assigning IP and MAC addresses. The workspace control tools operate with the Xen hypervisor and can also operate with KVM[5].
- **Workspace resource management**: is an open source solution to manage different VMs, but can be replaced by other technologies such as OpenNebula.
- **Workspace pilot**: is responsible for providing virtualization with few changes in cluster operation. This component handles signals and has administration tools.

4.3. OpenNebula

OpenNebula [OpenNebula Project 2010] is an open-source toolkit used to build private, public and hybrid clouds. It has been designed to be integrated with networking and storage solutions and to fit into existing data centers.

[5] http://www.linux-kvm.org/page/Main_Page

The OpenNebula architecture (Figure 3) is based on three basic technologies to enable the provision of services on a distributed infrastructure: virtualization, storage and network. All resource allocation is done based on policies.

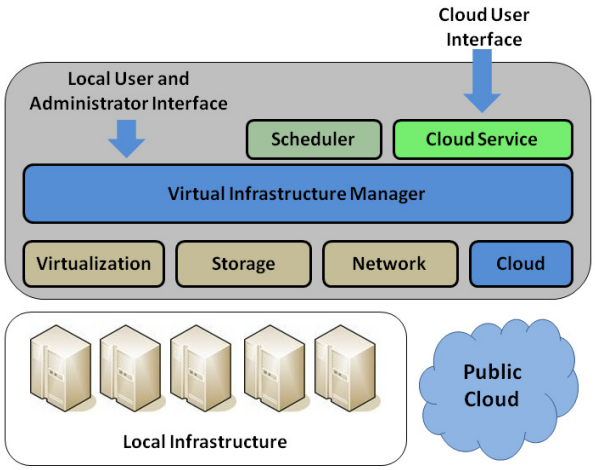


Figure 3. OpenNebula architecture [OpenNebula Project 2010]

The Cumulus Project [Wang et al 2008] is an academic proposal based on OpenNebula. Cumulus intends to provide virtual machines, virtual applications and virtual computing platforms for scientific applications. Visualizing the integration of already existing technologies, the Cumulus project uses HP and IBM blade serves running Linux and Xen hypervisor.

The Cumulus networking solution was called the "*forward*" mode, where users do not need to specify any network configuration information. Instead the backend servers are responsible for allocating a dynamic IP address for a VM and returning these to the users, making such networking solution transparent to the users.

The Cumulus design is a layered architecture (Figure 4) with three main entities: Cumulus frontend, OpenNebula frontend, and OS Farm. This proposal focuses on reaching scalability and autonomy of data centers.

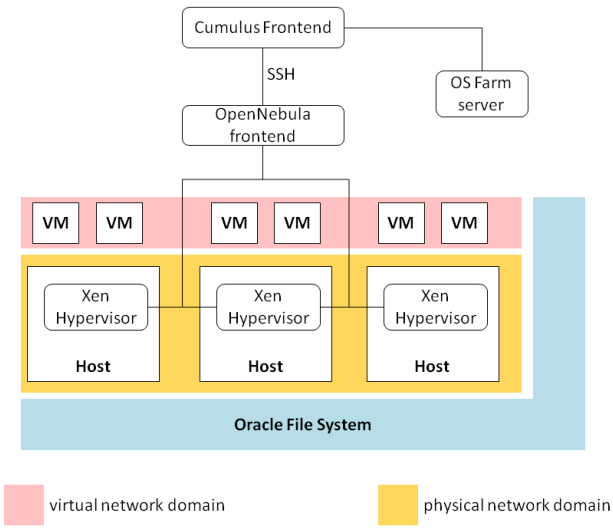


Figure 4. Cumulus architecture [Wang et al 2008]

- **Cumulus frontend**: the Cumulus frontend is the access point for a Cumulus system and is responsible for handling VM requirements.
- **OpenNebula frontend**: the OpenNebula frontend provides an interface to manage the distributed blade servers and the resources for VM deployment. To administrate a common user system, Cumulus uses NIS (Network Information System) and NFS (Network File System) to manage shared directory. Moreover, OpenNebula was merged with secure infrastructure solutions, such as LDAP (Lightweight Directory Access Protocol) and the Oracle Cluster File System.
- **OS Farm**: the OS Farm is a tool for VM template management that operates to generate and to store Xen VM images and virtual appliances.

4.4. Eucalyptus

Eucalyptus [Nurmi et al 2009] is an open source cloud computing framework focused on academic research. It provides resources for experimental instrumentation and study. Eucalyptus users are able to start, control, access and terminate entire virtual machines. In its current version, Eucalyptus supports VMs that run atop the Xen supervisor [Barham et al 2003].

According to [Nurmi et al 2009], the Eucalyptus project presents four characteristics that differentiate it from others cloud computing solutions: a) Eucalyptus was designed to be simple without requiring dedicated resources; b) Eucalyptus was designed to encourage third-party extensions through modular software framework and language-agnostic communication mechanisms; c) Eucalyptus external interface is based on the Amazon API (Amazon EC2) and d) Eucalyptus provides a virtual network overlay that both isolates network traffic of different users and allows clusters to appear to be part of the same local network.

The Eucalyptus architecture is hierarchical (Figure 5) and made up of four high level components, where each one is implemented as a stand-alone web service.

- **Node Controller (NC)**: this component runs on every node that is destined for hosting VM instances. An NC is responsible to query and control the system software (operating system and hypervisor) and for conforming requests from its respective Cluster Controller. The role of NC queries is to collect essential information, such as the node's physical resources (e.g. the number of cores and the available disk space) and the state of VM instances on the nodes. NC sends this information to its Cluster Controller (CC). NC is also responsible for assisting CC to control VM instances on a node, verifying the authorization, confirming resources availability and executing the request with the hypervisor.
- **Cluster Controller (CC)**: this component generally executes on a cluster front-end machine, or any machine that has network connectivity to two nodes: one running NCs and another running the Cloud Controller (CLC). A CC is responsible to collect/report information about and schedule VM execution on specific NCs and to manage virtual instance network overlay.
- **Storage Controller (Walrus)**: this component is a data storage service that provides a mechanism for storing and accessing virtual machine images and user

data. Walrus is based on web services technologies and compatible with Amazon's Simple Storage Service (S3) interface [Amazon 2006].

- **Cloud Controller (CLC)**: this component is the entry-point into the cloud for users. Its main goal is to offer and manage the Eucalyptus underlying virtualized resources. CLC is responsible for querying node managers for resources' information, making scheduling decisions, and implementing them by requests to CC. This component is composed by a set of web services which can be grouped into three categories, according their roles: resource services, data services, and interface services.

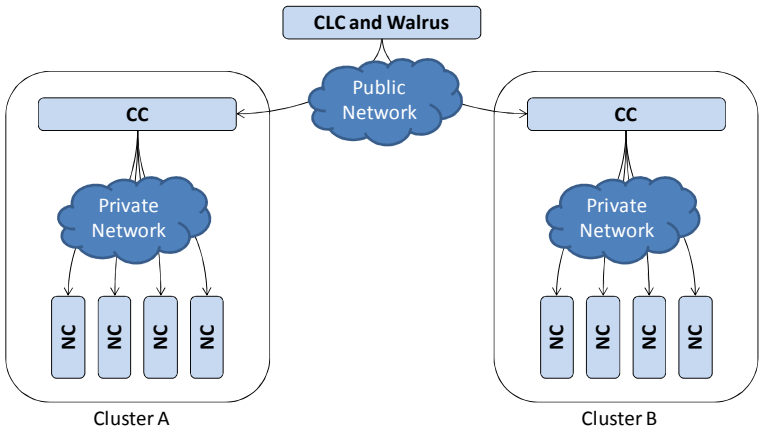


Figure 5. Eucalyptus architecture [Nurmi et al 2009]

Ubuntu Enterprise Cloud (UEC)[6] is an Amazon EC2 like infrastructure and is powered by Eucalyptus. Its main goal is to provide a simple process of building and managing internal infrastructure for cloud. The Ubuntu 9.04 Server Edition is integrated with Eucalyptus that uses the KVM hypervisor.

The UEC architecture is based on the Eucalyptus architecture in which each elements is an independent web service that publishes a Web Service Description Language (WSDL) document defining the API to interact with it.

Furthermore, UEC defines three layers for security: authentication and authorization, network isolation and Machine Instance Isolation (MInst). The authentication and authorization layer is responsible for locally generated X.509 certificates; the network isolation layer is important to prevent eavesdropping of network traffic and; the MInst layer consists of Networking isolation, Operating System isolation, and Hypervisor based machine isolation.

4.5. TPlatform

TPlatform [Peng et al 2009] is a cloud solution that provides a development platform for web mining applications, which is inspired in Google cloud technologies, and which acts as a Platform as a Service (PaaS) solution. Their infrastructure is supported by three technologies: a scalable file system called Tianwang File System (TFS) what is similar to the Google File System (GFS), the BigTable data storage mechanism, and the

[6] http://www.ubuntu.com/cloud

MapReduce programming model. The TPlatform framework is composed by three layers (Figure 6):

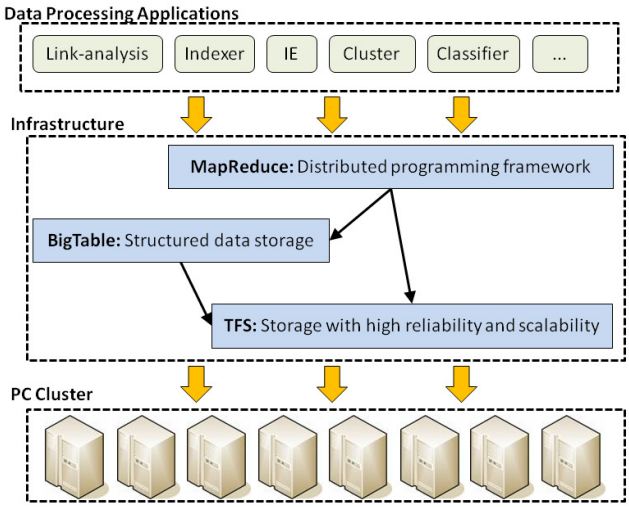


Figure 6. TPlatform framework [Peng et al 2009]

- **PC Cluster**: this layer provides the hardware infrastructure for data processing.
- **Infrastructure**: this layer consists of file system (TFS), distributed data storage mechanism (BigTable), and programming model (MapReduce).
- **Data Processing Applications**: this layer provides the services for users to develop their application (e.g. web data analysis and language processing).

4.6. Apache Virtual Computing Lab (VCL)

Apache VCL [VCL 2010] is an open-source solution for the remote access over the Internet to dynamically provision and reserve computational resources for diverse applications, acting as Software as a Service (SaaS) solution. VCL has a simple architecture formed by three tiers:

- **Web server**: represents the VCL portal and uses Linux/Apache/PHP solution. This portal provides an user interface that enable the requesting and management of VCL resources;
- **Database server**: storages information about VCL reservations, access controls, machine and environment inventory. It uses Linux/SQL solution;
- **Management nodes**: is the processing engine. A management node controls a subset of VCL resources, which may be physical blade servers, traditional rack, or virtual machines. It uses Linux/VCLD (perl)/image library solution. VCLD is a middleware responsible to process reservations or jobs assigned by the VCL web portal. According to type of environment requested, VCLD should assure that service (computational environment) will be available to user.

Figure 7 shows a conceptual overview of the VCL, where the user must connect firstly to the VCL Scheduling Application in order to access its resources through a web interface. Users may request a reservation to use the environment immediately or schedule to use it in the future.

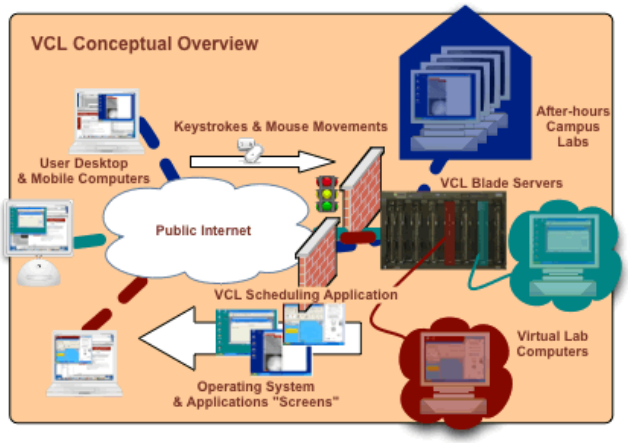


Figure 7. Apache conceptual overview [VCL 2010]

4.7. Enomaly Elastic Computing Platform

Enomaly ECP Community Edition under the AGPL license [Enomaly 2009] is the open source cloud solution offered by Enomaly Inc. This version focuses on virtual machine administration in small clouds environments. Compared with the Enomaly commercial solution (called Service Provider Edition), the Enomaly open source edition suffers from many restrictions, such as limited scalability, no capacity control mechanism, no support for accounting and metering, and so on.

5. Discussion

As pointed out earlier in this paper, there are several solutions for cloud computing[7] focusing on different areas and ranging from hardware resource outsourcing to user services providing. Each solution presents a different vision about cloud architecture and implementation. Moreover, each approach has an implication that directly impacts its business model: the closer to the hardware level, the more options a user can handle but at the cost of having to configure her cloud (more configuration flexibility).

Amazon EC2 and IBM Capacity on Demand (CoD) are solutions that offer to their users this configuration flexibility. In this business model, users can choose and configure computational resources at the hardware level and OS levels. At the other extreme, solutions like Google App Engine and Windows Azure, try to turn development easy to their users, but at the same time, confine them to specific APIs and software platforms. Moreover, solutions like JoliCloud[8] are more limited as they offer a single service (operating system). In the middle, there are solutions that offer a middleware-like approach to users, where the hardware resources can be configured and handled subject to some restrictions and where applications can also be developed.

All the presented open-source solutions and the cited commercial solutions are categorized into Figure 8. The graphic compares solutions and their business model (hardware, middleware and user level) according to configuration flexibility.

---

[7] More than 500 Cloud solutions have been reported.

[8] http://www.jolicloud.com/

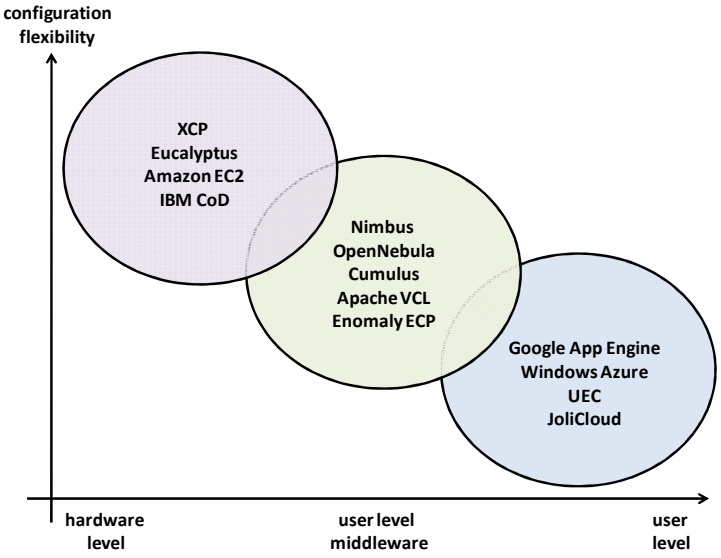


Figure 8. Cloud computing solutions

Finally, Table 1 presents a comparative board of the open source cloud solutions described in this paper, in terms of the service type (IaaS, PaaS, and SaaS), the main characteristics, and the infrastructure technologies. The table also cites some users of each cloud solution.

Table 1. Comparison between open-source Cloud Computing solutions

Solutions	Service	Main Characteristics	Infrastructure	Used by
XCP	IaaS	Only a tool for automatic maintenance of clouds	Xen	XCP community
Nimbus	IaaS	Aims to turn legacy clusters into IaaS Clouds	Xen hypervisor and KVM	Brookhaven National Labs[9]
OpenNebula	IaaS	Policy-driven resource allocation	Xen hypervisor	Cumulus Project
Eucalyptus	IaaS	Hierarchical Architecture	Xen hypervisor and KVM	UEC
TPlatform	PaaS	Focus on web text mining applications	TFS, BigTable and MapReduce	TPlatform Project[10]
Apache VCL	SaaS	Internet access for several applications	VMware	Educational[11] and Government[12] users
Enomaly	IaaS	Open version is focused in small clouds	Xen, KVM and VirtualBox	Several companies[13]

[9] http://www.bnl.gov/rhic/default.asp

[10] http://net.pku.edu.cn/~webg/tplatform/

[11] East Carolina University, Johnston Community College, North Carolina Central University, University of North Carolina at Greensboro, Wake Technical Community College and Western Carolina University

[12] North Carolina Community College System

[13] Some companies that use Enomaly Service Provider Edition: Orange/France Télécom, Bank of China, City Network, CentriLogic

6. Conclusions and Future Works

There is a clear need for the standardization of current cloud platforms at least of terms of interface, negotiation and access through Web services. Understandably, this is a considerable task as many clouds use different abstraction levels, some are generic whereas others focus on a specific application domain, etc. Some initial steps have been taken into this direction with the setup of the Open Cloud Manifesto, an initiative supported by hundreds of companies. In the mean time we will continue to see some clouds such as Nimbus implementing a number of front ends (e.g. Amazon EC2 and WSRF) to ensure access to their existing users.

Interestingly, some solutions such as the OpenNebula have been first to adopt policies for resource management. The use of policies remains a challenge in many areas and clouds may benefit from it. It is also important to acknowledge the leadership and string presence of academic efforts such as Eucalyptus and Xen. These have been at the forefront of inovation supported by the many commercial cloud systems that currently are based on these. These efforts are expected to continue as more work is needed to remove much of the misticism and conflicts surrounding the use of clouds.

As future work, authors proposes a quantitative comparison of the presented solutions through performance evaluation measurements.

References

Amazon. (2006) “Amazon simple storage service (Amazon S3) – API Reference”, http://docs.amazonwebservices.com/AmazonS3/2006-03-01/.

Andersen, D. (2002) “Theoretical Approaches to Node Assignment”, Unpublished manuscript. http://www.cs.cmu.edu/dga/papers/andersen-assign.ps

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M. (2009) “Above the Clouds: A Berkeley View of Cloud Computing”, Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A. (2003) “Xen and the art of virtualization”. In: 19th ACM symposium on Operating systems principles, New York, NY, USA.

Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I. (2009) “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”. In: Future Generation Computer Systems, Elsevier B. V.

Chowdhury, N.M. M. K. and Boutaba, R. (2010) “A survey of network virtualization”. In: Computer Networks, volume 54, issue 5, pages 862-876. Elsevier B. V.

Citrix Systems. (2010) “Xen Cloud Platform - Advanced Virtualization Infrastructure for the Clouds”, http://www.xen.org/products/cloudxen.html.

Citrix Systems. (2010) “Xen Hypervisor”, http://xen.org/products/xenhyp.html.

Citrix Systems. (2010) “The Xen Cloud Project”, The Citrix Blog. Available at http://community.citrix.com/pages/viewpage.action?pageId=81690805

Enomaly. (2009) “Enomaly – Elastic Computing”. http://src.enomaly.com/.

Keahey, K. (2009) “Nimbus: Open Source Infrastructure-as-a-Service Cloud Computing Software”, Workshop on adapting applications and computing services to multi-core and virtualization, CERN, Switzerland.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J. (2008) “OpenFlow: enabling innovation in campus networks”, ACM SIGCOMM Computer Communication Review.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L. and Zagorodnov, D. (2009) “The Eucalyptus Open-Source Cloud Computing System”. In: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid.

OpenCloud. (2009) “The Open Could Manifesto”. http://www.opencloudmanifesto.org/.

OpenCloud. (2010) “Cloud Computing Use Cases Whitepaper - Version 3.0”. http://groups.google.com/group/cloud-computing-use-cases?hl=en.

OpenNebula Project. (2010) “OpenNebula.org – The Open Source Toolkit for Cloud Computing”, http://www.opennebula.org/.

Peng, B., Cui, B. and Li, X. (2009) “Implementation Issues of A Cloud Computing Platform”. IEEE Data Engineering Bulletin, volume 32, issue 1.

Vaquero, L.M., Merino, L. R. Caceres, J. Lindner, M. (2008) “A break in the clouds: towards a cloud definition”. ACM SIGCOMM Computer Communication Review.

VCL. (2010). “Virtual Computing Lab”. https://cwiki.apache.org/VCL/.

Vouk, M.A. (2008) “Cloud Computing – Issues, Research and Implementations”. In: Journal of Computing and Information Technology. University of Zagreb, Croatia.

Xen.org. (2009) “Xen Cloud Platform Administrator’s Guide: Release 0.1”.

Wang, L., Tao, J., Kunze, M., Rattu, D. and Castellanos, A. (2008) “The Cumulus Project: Build a Scientific Cloud for a Data Center”. In: Workshop on Cloud Computing and Its Applications, Chicago, USA.

Uma análise de recursos virtualizados em ambiente de HPC

Thais C. de Mello[2], Bruno Schulze[1], Raquel C. G. Pinto[2], Antonio R. Mury[1]

[1]Coordenação de Ciência da Computação - Laboratório Nacional de Computação Científica
Av. Getúlio Vargas 333, Quitandinha, Petrópolis, RJ.

[2]Sistemas e Computação - Instituto Militar de Engenharia
Praça General Tiburcio 80, Urca, Rio de Janeiro, RJ

{thaiscm, schulze, aroberto}@lncc.br, raquel@ime.eb.br

Abstract. *How the cloud paradigm can support the solution of High Performance Computing (HPC) problems has not been fully answered yet. This raises a number of questions and, among them, the effect of resources virtualization in terms of performance. We present the analysis of the use of virtual clusters and the factors considered crucial to the adoption of such solution, mainly on infrastructure development to facilitate their creation and use of a custom environment for education, training, testing or development.*

Resumo. *A questão de quanto o uso do paradigma de nuvem pode servir de apoio a solução de problemas de Computação de Alto Desempenho (HPC) ainda não foi totalmente respondida. Suscita uma série de questionamentos e, dentre eles, o efeito da virtualização de recursos em termos de desempenho. Neste artigo é feita uma análise do uso de clusters virtuais e os fatores considerados determinantes para adoção desse tipo de solução, principalmente no desenvolvimento da infraestrutura que facilite sua criação e o uso de um ambiente personalizado para fins educativos, de treinamento, de teste ou de desenvolvimento.*

1. Introdução

O desenvolvimento científico e tecnológico observa uma crescente necessidade e dependência de recursos computacionais de alto desempenho e capacidade de armazenamento. Isso se deve ao fato de que o desenvolvimento das atividades de pesquisas nas diversas áreas da ciência vem gerando um conjunto de dados cada vez maior e mais complexo, tornando o uso da computação de alto desempenho imprescindível para que a análise dos dados, seu armazenamento e compartilhamento sejam feitos de forma rápida e eficiente.

A utilização da computação intensiva, como acima descrito, para obtenção de resultados científicos, caracteriza o termo e-Science. Tal conceito deve permitir o acesso aos recursos computacionais distribuídos, ao armazenamento, ao compartilhamento de dados e resultados, viabilizando o desenvolvimento de pesquisas colaborativas de forma global e a grandes distâncias.

Entretanto, torna-se um grande desafio prover um ambiente de *hardware* e *software* que ofereça a seus usuários uma abstração sobre os recursos computacionais necessários para execução de simulações. Similarmente, prover uma infraestrutura de

gerência dos elementos computacionais utilizados e produzidos durante essas simulações também se mostra um processo complexo. Além disso, o avanço científico exige um investimento crescente em equipamentos e infraestrutura, de forma a estar em sintonia com as necessidades de tal avanço.

Uma alternativa tecnológica adotada pelos centros de e-Science foi o uso de Computação em *Grid*, que permite a utilização de diversos recursos computacionais heterogêneos e dispersos por várias instituições. Essa característica possibilita que seus usuários compartilhem recursos de forma confiável, consistente e transparente. Contudo, a experiência obtida em diversos projetos concebidos a partir da infraestrutura de *Grid* mostrou alguns problemas, tais como instalação, configuração e usabilidade complexas, além da necessidade de controles administrativos intensivos, o que dificultou sobremaneira o aumento dessa infraestrutura em termos de uso e espaço. Ainda assim, o *Grid* permitiu um avanço em termos de computação utilitária no qual a estratégia é baseada no encapsulamento da infraestrutura computacional na forma de serviços.

Nesse ínterim houve o desenvolvimento de novas tecnologias tanto em termos de capacidade de processamento, quanto em capacidade de redes de comunicação, o que possibilitou um avanço no uso mais efetivo e compartilhado dos recursos computacionais, principalmente pelo desenvolvimento de processadores dedicados à virtualização.

Por conseguinte, esse avanço da tecnologia de virtualização possibilitou que, a partir de uma única plataforma física, pudessem coexistir várias plataformas sendo executadas ao mesmo tempo e de forma isolada. Com isso, é possível ter, em um mesmo equipamento físico, vários ambientes virtuais diferentes, cada um com seus respectivos sistemas e aplicações.

A virtualização também torna possível a criação de um ambiente dinâmico e flexível, que aproveita o máximo dos recursos computacionais ociosos por meio de seu compartilhamento, diminuindo a necessidade de adquirir novos equipamentos. Com o reaproveitamento dos recursos já existentes, facilita-se o suporte e a manutenção, permitindo a existência de um número maior de plataformas virtuais sem ter o respectivo aumento no número de plataformas reais.

Esses avanços propiciaram o surgimento de um novo paradigma da computação, chamado de Computação em Nuvem. Tal tecnologia se propõe a ser um sistema de computação distribuída em larga escala, oferecendo um conjunto de recursos virtualizados, dinamicamente escaláveis, gerenciáveis em termos de capacidade de processamento, armazenamento, número de plataformas e serviços. Tem como princípio a oferta de serviços adequados às necessidades do usuário, de maneira a reforçar o conceito de elasticidade sob o ponto de vista da demanda de recursos, cuja alocação e desalocação ocorre em função da necessidade, tornando desnecessária a imobilização de um grande capital para sua aquisição.

A estrutura de computação em nuvem, que passaremos a tratar por *Cloud*, tem como características flexibilidade, escalabilidade e portabilidade. A flexibilidade se dá pelo fato da infraestrutura de TI poder ser adaptada em função do modelo de negócios vigente, sem que estes estejam necessariamente atrelados à infraestrutura passada. A escalabilidade se caracteriza pelo aumento e pela redução de recursos em função das necessidades, possibilitando uma resposta rápida à demanda. A portabilidade, por sua

vez, se dá pela dissociação entre o espaço geográfico e os recursos e as pessoas que os utilizam, permitindo que tais recursos estejam espalhados ou agregados em sistemas especializados, com a consequente redução de custo em função da economia de escala.

O conceito acima descrito tem eco no termo *Virtual Workspace*, que pode ser compreendido como um ambiente de execução configurável e gerenciável de forma a atender aos requisitos dos seus usuários, definindo, para tanto, quais os requisitos de *hardware*, configuração de *software*, isolamento de propriedades e outras características. Com base nesse conceito e no de *Cloud*, o presente artigo se propõe a apresentar a análise de um conjunto de testes que foram realizados para verificar a viabilidade do desenvolvimento de uma infraestrutura cuja essência é facilitar a criação de ambientes de *clusters* virtuais.

O objetivo é possibilitar que seus usuários possam criar e utilizar um ambiente personalizado para fins educativos, treinamento, teste ou desenvolvimento. Para tanto, eles apenas configurarão características básicas, sem que precisem deter conhecimentos técnicos sobre a estrutura, manutenção ou configuração do ambiente. Esse ambiente tem como principais características: a flexibilidade, ao possibilitar que o usuário configure e seja capaz de utilizar várias estruturas de *clusters* em um mesmo conjunto de recursos homogêneos ou não; a escalabilidade, na medida que é possível aumentar ou reduzir o número de nós em função de suas necessidades ou restrições do ambiente; e a portabilidade, ao permitir que o ambiente criado seja armazenado tanto para uso futuro quanto para ser utilizado em outra estrutura disponível, dissociado do espaço físico em que se encontra.

Este trabalho está estruturado como apresentado a seguir: Na seção 2 são apresentados os trabalhos relacionados ao uso de *clusters* virtuais, virtualização de recursos, *Grid* e *Cloud*. Na seção 3 são apresentados os objetivos dos testes realizados, equipamentos, aplicações e a metodologia de análise. Na seção 4 é feita a análise dos resultados e na seção 5 são apresentadas as conclusões.

2. Trabalhos Relacionados

O uso de computação intensiva em apoio à pesquisa e ao processamento de grande quantidade de dados científicos exige utilização colaborativa dos recursos. A infraestrutura para o tratamento desse tipo de informação e as ferramentas para auxiliarem nessa tarefa devem possibilitar que seus usuários (e-scientists) as utilizem de forma transparente. A otimização no uso dos recursos em grandes centros de HPC e sua integração com a infraestrutura de e-Science apresentam desafios, destacando-se a necessidade de algoritmos mais efetivos para o processamento da informação e o uso racional dos recursos, caracterizados pela necessidade de ferramentas e códigos mais eficientes e escaláveis, capazes de melhor aproveitar o crescente aumento do número de núcleos em um único processador [Riedel et al. 2007].

A experiência obtida com o uso do *Grid* serviu de base para um modelo de computação distribuída mais flexível e adequado às necessidades específicas dos usuários. Além disso, permitiu um avanço em termos de computação utilitária e, ao incorporar a arquitetura aberta de serviços de *Grid* (OGSA), motivou o aparecimento de uma nova estratégia baseada no encapsulamento e na oferta de serviços. Essa abordagem reforça o conceito de elasticidade na demanda de recursos computacionais, em que a variação dessa necessidade é acompanhada pela correspondente variação na alocação. Apesar da

tecnologia de Grid poder ser utilizada em uma ampla variedade de aplicações comerciais e de seu sucesso na área acadêmica, sua evolução não alcançou plenamente os benefícios inicialmente pretendidos - a escalabilidade, o fato de ser centrada nos usuários, a facilidade de uso, configuração e gerenciamento [Stanoevska-Slabeva et al. 2009].

Surge assim, como resposta a essas necessidades, o modelo de *Cloud*. Este se caracteriza por ser um modelo econômico e de negócios que apresenta grande flexibilidade de recursos, permitindo que até mesmo as pequenas corporações possam ser capazes de se manterem atualizadas e competir em condição de igualdade com corporações maiores [Stanoevska-Slabeva et al. 2009].

Segundo [Foster et al. 2008] tanto *Grid* quanto *Cloud* procuram reduzir o custo da computação, aumentando a confiabilidade, a flexibilidade e transformando o conceito de computação de "algo" que se compra e que nós operamos em "algo" operado por terceiros. Ambas as infraestruturas têm as mesmas necessidades, tanto em termos de gerenciamento de recursos, quanto pelo fato de permitirem que seus usuários possam utilizá-los em toda a sua capacidade. Contudo, a evolução que o *Cloud* representa está na mudança do foco de uma infraestrutura que fornece recursos computacionais para uma baseada em economia de escala, fornecendo recursos/serviços de forma mais abstrata a seus usuários.

Com base no exposto anteriormente, é de grande importância atentar para o conceito de *Virtual Workspace* apresentado por [Keahey et al. 2005], que consiste em disponibilizar de forma automática recursos e prover o ambiente de execução que atenda às necessidades do usuário. Percebe-se que uma das principais ferramentas dos cenários acima descritos é a virtualização de recursos, que apesar de ser um conceito utilizado há bastante tempo, teve suas funcionalidades ampliadas e seu uso difundido nos últimos anos. Tal fato se deve ao desenvolvimento tecnológico alcançado sobretudo pelos avanços na arquitetura de processadores e pelo aumento da capacidade de rede. No caso dos processadores, houve não só o aumento da capacidade de processamento, mas também a implementação de instruções dedicadas à virtualização de recursos.

Em sintonia com a evolução tecnológica, vários aspectos motivaram e aceleraram o uso da virtualização nos últimos anos, tais como: a capacidade de consolidação de servidores, inclusive auxiliando a continuidade na utilização de sistemas legados; a confiabilidade, padronização e a velocidade na disponibilização de novos recursos/plataformas; a atualização, tanto de *hardware*, quanto de aplicações sem a necessidade de interrupções em serviços e servidores; e a segurança, por permitir a configuração de ambientes de testes, desenvolvimento e treinamento sem o comprometimento direto dos recursos em produção [Dittner and Rule 2007]. Há, no entanto, algumas limitações que ainda persistem, podendo ser citadas: a necessidade de um gerenciamento cuidadoso do nível de comprometimento/carga das máquinas hospedeiras; a correta avaliação e controle do que pode ser executado/transposto para uma máquina virtual; e a mudança de paradigma na administração e organização dos recursos virtualizados.

Em [Mergen et al. 2006], argumenta-se que o uso da virtualização em HPC possibilita tanto a configuração dedicada dos sistemas quanto a manutenção de compatibilidade com sistemas legados, ou seja, a coexistência nos mesmos recursos. A extensão do número de máquinas virtuais permite, por exemplo, que o Monitor de Máquinas Virtuais (VMM) apresente, para um *cluster* virtual, a ideia da existência de inúmeros nós virtuais

em poucos nós reais, permitindo que um teste em escala real de aplicações tipo MPI possa ser feito mesmo na presença de poucos recursos, ainda que estes estejam sendo utilizados por outras aplicações.

Os autores defendem, ainda, que o custo da virtualização em si e o benefício sob o ponto de vista do desempenho que ela realmente oferece são fatores que devem ser levados em consideração. Levanta o questionamento quanto aos requisitos necessários ao desenvolvimento de um VMM específico para o uso em HPC (tamanho das páginas de memória, escalonamento dos recursos virtualizados, mecanismos de DMA e alto desempenho em termos de I/O, balanceamento de carga, preempção e migração).

Em [Evangelinos and Hill 2008] temos a análise de um Provedor/Sistema de *Cloud* (Amazon EC2), utilizado para a configuração de um *cluster* virtual. Os autores avaliaram que é viável esse tipo de solução, principalmente quanto à facilidade de uso e à disponibilização de sistemas que meçam seu impacto nos paradigmas de HPC. Assim sendo, constataram que a solução de recursos "sob-demanda", associados a customização com foco na solução de um problema/cenário específico e gerenciados no momento desejado, é vantajosa mesmo que com perda de desempenho.

[Ranadive et al. 2008] argumentam que a virtualização tem se mostrado uma ferramenta eficaz para o tratamento de falhas em equipamento, aumentando a portabilidade das aplicações e auxiliando a descoberta e correção de falhas em algoritmos complexos. Todavia, pesquisadores mostram-se relutantes em utilizá-la em aplicações de HPC. Apontam, nesse sentido, dois fatores: o primeiro se deve em parte ao desejo de quererem explorar todas as capacidades dos recursos e plataformas; o segundo, considerado o mais importante, é a resistência quanto ao compartilhamento de recursos e seu efeito na degradação do nível de desempenho.

Fatores também apontados por [Gavrilovska et al. 2007] incluem: a determinação efetiva do nível de perda aceitável introduzida pela virtualização; a solução de problemas de inconsistência em relação ao *hardware* ou aplicações e o impacto do uso das plataformas com multiplos núcleos. [Ranadive et al. 2008] conclui que, dependendo do tipo de interação entre as aplicações e o nível de comunicação (I/O) efetuado na solução de aplicações utilizando MPI, mostra-se mais vantajoso associar cada máquina virtual a um núcleo, como componentes separados dentro de uma mesma plataforma.

[Brandt et al. 2009] alerta que, apesar das promessas propagadas pelo uso de *Cloud*, para um ambiente de HPC é necessária a existência de uma gerência que auxilie o usuário tanto na escolha dos recursos a serem alocados quanto na análise a posteriori. Essa análise tem por fim otimizar este ambiente virtual em utilizações futuras, devendo ser capaz de responder a parâmetros adicionais e ajustá-los dinamicamente mesmo em tempo de execução. De forma análoga, ela deve expor os parâmetros de avaliação e essas devem ser realizadas nos recursos virtuais e reais, possibilitando aos provedores de recursos/serviços oferecer um melhor nível de qualidade de serviços a seus usuários.

Nos trabalhos supra-relacionados, em relação à possibilidade do uso de recursos virtualizados para o oferecimento de serviços de alto desempenho aos usuários, destacamos os seguintes itens importantes: a necessidade de que esse ambiente seja flexível, portável e elástico; o desenvolvimento de mecanismo de avaliação dos recursos reais e virtuais para a otimização do seu uso; o impacto do compartilhamento de recursos; o de-

senvolvimento de mecanismos de balanceamento de carga de forma a minimizar a perda de desempenho e o desenvolvimento de ferramentas próprias para auxiliar os usuários na configuração e gerenciamento dos recursos desse ambiente.

Este trabalho insere-se em um projeto em andamento que objetiva verificar a viabilidade do desenvolvimento de uma infraestrutura que facilite a criação de *clusters* virtuais. Pretende-se, assim, possibilitar que os usuários sejam capazes de criar e utilizar um ambiente personalizado, recuperável, escalável e de fácil uso e configuração, seja para fins educativos, de treinamento, seja para teste ou desenvolvimento, de forma a otimizar o uso de equipamentos existentes em um ambiente de Grid, podendo, porém, tal infraestrutura ser aplicada em outros ambientes distribuídos. Na seção a seguir será apresentada a análise de um conjunto de testes que buscam avaliar o impacto do uso da virtualização em relação ao desempenho (carga de processamento e de comunicação) em um ambiente de HPC com um estudo de caso na utilização de *clusters* virtuais.

3. Objetivos e Métodologia de Análise

Nesta seção será apresentada a metodologia de análise utilizada para verificar não só o impacto do uso de *clusters* virtuais em um ambiente distribuído, mas também o efeito da concorrência no uso destes recursos. Os testes a seguir buscam determinar os valores de desempenho em termos de capacidade de processamento (Gflops) e capacidade de comunicação (MBs/s), utilizando dois pacotes de testes de *benchmarks* para sistemas paralelos ou distribuídos.

3.1. Objetivos

Os seguintes objetivos abaixo foram utilizados na estruturação dos testes realizados:

1. Determinar a perda de desempenho de um *cluster* virtual em comparação a um *cluster* real, com o uso do VMM escolhido para o trabalho;
2. Determinar o efeito do tipo de distribuição *Linux* utilizada em conjunto com o VMM; e
3. Determinar o efeito do uso concorrente dos recursos por diversos *clusters* virtuais.

3.2. Infraestrutura de equipamentos

Os experimentos foram realizados em 6 servidores, sendo que cada um contava com as seguintes características: dois processadores Xeon 5520 Quad Core (2.26GHz), com 12 GB de memória RAM DDR3 (1333Mhz), com tecnologia de *Hyperthread* e Instruções de Virtualização habilitadas e Rede Gigabyte.

3.3. Ferramenta de Cluster

O Pelican HPC [Creel 2008] foi utilizado na criação e no gerenciamento tanto do *cluster* real quanto do virtual. Ele tem como principal característica o fato de poder ser executado totalmente em RAMDISK (o SO trata a memória alocada como se fosse um disco rígido, dispensando o uso do disco real), tornando, assim, a montagem do *cluster* mais rápida. Sua instalação ocorre a partir de um *Live-CD* que possui as bibliotecas LAM-MPI e OpenMPI (32 e 64 bits). Os nós de trabalho são carregados via PXE (padrão de *boot* remoto Intel, que utiliza a rede para o carregamento das aplicações necessárias, permitindo, assim, estações sem disco) e usam o nó frontal como mestre.

3.4. Virtualizador

Utilizou-se um sistema de Virtualização Total [SUNMicrosytems 2010] que permite que um SO convidado possa ser executado, sem ser modificado, sobre o SO hospedeiro. Esse sistema de virtualização tem suporte a instruções de virtualização *Hardware-Assisted Intel Virtualization Technology* e suporta, no caso de Plataformas Hospedeiras (32/64 bits), Solaris, Linux, Mac OS X, Windows. Já no caso de Plataformas Convidadas (32/64 bits), *Solaris*, *Linux*, BSD, IBM-OS2 e *Windows*. Além disso, tem suporte a *Open Virtualization Format* (OVF), permitindo a importação e exportação de *aplliances* com outros tipos de virtualizadores e suporte a *Symmetric Multiprocessing* (32 núcleos).

3.5. Pacotes de Teste de Desempenhos

Para medir o desempenho dos *clusters* foram utilizados duas aplicações de *benchmarks*: a primeira com o objetivo de avaliar a capacidade de transmissão de dados (Beff - MB/s) e a segunda avalia a capacidade de processamento (PARPAC - Gflops)[Kredel and Merz 2004].

1. Beff - mede a carga acumulada de comunicação de sistemas paralelos ou distribuídos. Para os testes são utilizados diversos tamanhos de mensagens, padrões de comunicação e métodos. O algoritmo usa uma média que leva em consideração mensagens longas e curtas que são transferidas com diferentes valores de largura de banda em aplicações reais. O cálculo é feito levando-se em conta o número de processos MPI e o tamanho das mensagens. Os padrões são baseados em anéis com distribuições aleatórias. A comunicação é implementada de três diferentes maneiras com o MPI e para cada medida é usada a banda máxima.
2. PARPAC - simula a dinâmica de um fluido (lattice Boltzmann), calculando a permeabilidade da estrutura e tendo como retorno o cálculo de desempenho em Gflops. O código é escrito totalmente em C++ e de forma paralelizada, capaz de fazer a decomposição automática do domínio e o balanceamento de carga, com otimização da comunicação entre os nós calculados.

3.6. Tipos de Testes e Medidas realizadas

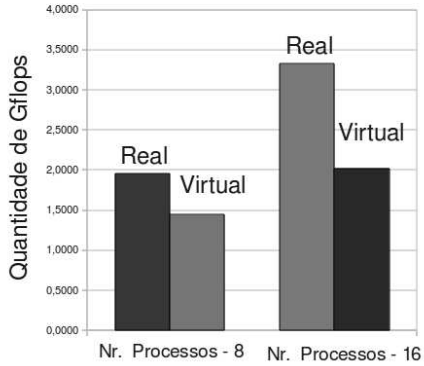
Para o cálculo dos resultados foram tomadas 30 medidas para cada teste, apresentando-se o cálculo de sua média e o intervalo de confiança.

1. Teste 1 - fez-se a medida de desempenho do *cluster* real, sendo este instalado e configurado com um servidor como *frontend* e cinco servidores como nós de trabalho. Esse teste teve como objetivo verificar o desempenho do *cluster* real instalado para servir de comparação com o Teste 2.
2. Teste 2 - inicializou-se uma máquina virtual em cada servidor e foi instalado o *cluster*. Uma máquina virtual foi utilizada como *frontend* e as outras cinco máquinas virtuais como nós de trabalho (cada máquina virtual foi configurada como possuindo 8 núcleos e 3546MB de memória). O teste 2 teve como objetivo medir o desempenho do *cluster* virtual. A opção de se utilizar um *frontend* com a mesma capacidade dos nós de trabalho se deve ao fato que ambos os pacotes de teste podem utilizar o *frontend* para o balanceamento de carga (inclusive esse uso foi notado durante os testes).

3. Teste 3 - 5 máquinas virtuais foram inicializadas em um único servidor (cada máquina virtual com 2 núcleos e 2048 MB de memória), uma máquina como *frontend* e quatro máquinas como nós de trabalho. O objetivo desse teste foi medir o efeito do tipo de distribuição *Linux* no ambiente de *cluster* virtual, ou seja, o impacto que o conjunto de bibliotecas que caracterizam uma distribuição tem em relação ao virtualizador, assim como a comparação entre o efeito da arquitetura de 32 Bits e de 64 Bits.
4. Teste 4 - criaram-se 3 *clusters* virtuais em um único servidor, cada um com 3 máquinas virtuais (2 núcleos e 1024 MB de memória), sendo uma como *frontend* e as outras duas como nós de trabalho. Cada *cluster* foi inicializado separadamente para verificar o impacto no desempenho pela concorrência nos recursos. O teste 4 teve como objetivo ver o efeito da concorrência no caso de existirem mais de um *cluster* sendo executado em um mesmo servidor.

4. Análise dos Resultados

O testes 1 e 2 foram realizados com a finalidade de comparar o desempenho do *cluster* real com o *cluster* virtual. Para a realização dos testes foi utilizado a distribuição *Linux* 64 Bits padrão do Laboratório de Computação Científica Distribuída - ComCiDis - LNCC. As medidas obtidas encontram-se apresentadas nas figuras 1 e 2. A Figura 1 apresenta o resultado em relação à carga de processamento e podemos notar que, no caso de 8 processos, o *cluster* virtual teve um desempenho de 73,7% em comparação com o *cluster* real. Para o caso de 16 processos, o desempenho caiu para 60,8%. A tabela da figura apresenta os valores obtidos tanto para o *cluster* real quanto para o virtual em termos de Gflops, com um intervalo de confiança de 95%.



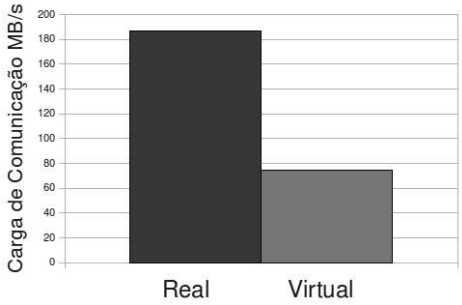
	Real 8 Processos	Virtual 8 Processos	Real 16 Processos	Virtual 16 Processos
Média	1,956	1,442	3,323	2,023
Intervalo Inferior	1,954	1,441	3,321	2,022
Intervalo Superior	1,959	1,443	3,326	2,024

Figura 1. Desempenho da capacidade de processamento entre o cluster real e o cluster virtual

Para o teste de carga de comunicação apresentado na (Figura 2) o desempenho alcançado pelo *cluster* virtual em relação ao *cluster* real foi de 39,8%. Este baixo desempenho tem como consequência o uso da virtualização total. A fim de permitir uma menor dependência da máquina hospedeira, aumentando a portabilidade, é criada uma camada de dispositivos I/O virtuais. No caso específico desse teste, é necessário que o sistema convidado (máquina virtual) escreva os dados na parte da memória virtual alocada para tanto. Porém, ao tentar fazê-lo, o VMM intercepta a tentativa do sistema convidado e

mapeia para o local de memória real, onde está mapeado o dispositivo de I/O real. O VMM tem que manter um controle e mapear todos os endereços de memória dos dispositivos virtuais e encaminhá-los para os dispositivos reais correspondentes. Tal fato causa uma sobrecarga que diminui significativamente o desempenho.

A solução atualmente encontrada para diminuir esta perda é o uso de bibliotecas "Virtio". Essas surgem como alternativas à proposta adotada pelo XEN na tentativa de fornecer uma série de *drivers* padrões para distribuições *Linux*, sendo que tais drivers são adaptáveis a vários tipos de VMM, não adicionando sobrecarga. O primeiro a usar esse tipo de solução foi o KVM [Russell 2008], [Nussbaum et al. 2009] e, apesar de o virtualizador utilizado neste trabalho já possuir suporte a estes tipos de biblioteca na sua última versão, não foi objeto deste trabalho, sendo programado para trabalhos futuros.



Carga de Comunicação MB/s

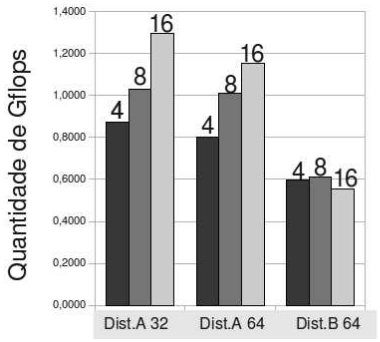
Real	Virtual
187,19	74,64
186,47	72,64
188,11	76,65

Figura 2. Desempenho carga de comunicação do cluster real com o cluster virtual

O teste 3 teve como objetivo ver o impacto do tipo de distribuição *Linux* no desempenho dos *clusters* virtuais. A Figura 3 apresenta a comparação do desempenho entre *clusters* virtuais utilizando como SO básico duas distribuições *Linux* diferentes, porém, com o mesmo pacote básico (chamaremos de distribuição A e B). Fez-se ainda uma comparação dentro da mesma distribuição utilizando SO de 32 bits e SO de 64 bits (Distribuição A32 e A64). A Figura 3 nos mostra que, à medida que a carga de processos foi aumentando na distribuição A32, sua capacidade de processamento também aumentou (de 0,87 Gflops com 4 processos a 1,3 Gflops com 16 processos).

A mesma distribuição no caso de 64 bits (A64) apresentou similarmente uma resposta positiva (de 0,80 Gflops com 4 processos para 1,2 Gflops com 16 processos). Percebe-se que a diferença entre ambas é praticamente nula, sendo que o motivo da diferença em favor da distribuição A32 se deve ao fato da distribuição A64 conter bibliotecas de 32 Bits, o que diminui ligeiramente o seu desempenho. Porém, a distribuição B64 apresentou um desempenho de 74,5% em relação à distribuição A64 com 4 processos, caindo para 48% com 16 processos. É importante notar que sua capacidade de processamento inicialmente subiu na mudança de 4 para 8 processos (0,60 Gflops para 0,61 Gflops - Figura 3 Distribuição B 64), porém caiu quando submetida a 16 processos (0,55 Gflops).

O motivo para o baixo desempenho em relação a outra distribuição e a queda, no caso de 16 processos, está relacionada à incapacidade que se notou em se distribuir igualmente o processamento das máquinas virtuais entre todos os núcleos da máquina real. A Figura 4 apresenta uma amostra da carga entre os núcleos da Distribuição



		Distribuição A32	Distribuição A64	Distribuição B64
Média	4 processos	0,8712	0,8032	0,5980
	8 Processos	1,0287	1,0113	0,6121
	16 Processos	1,2977	1,1550	0,5548
Limite Inferior	4 processos	0,8711	0,8031	0,5975
	8 Processos	1,0281	1,0113	0,6121
	16 Processos	1,2971	1,1550	0,5544
Limite Superior	4 processos	0,8713	0,8033	0,5984
	8 Processos	1,0292	1,0113	0,6122
	16 Processos	1,2982	1,1550	0,5555

Figura 3. Desempenho de distribuições Linux - capacidade de processamento

A64 e da Distribuição B64. No caso da capacidade de processamento esta diferença se deve à política de distribuição de recursos de cada uma delas, que em função do uso poderá se mostrar vantajosa ou não, neste caso a liberdade na distribuição de carga pelos núcleos/threads, beneficiou a distribuição A64.

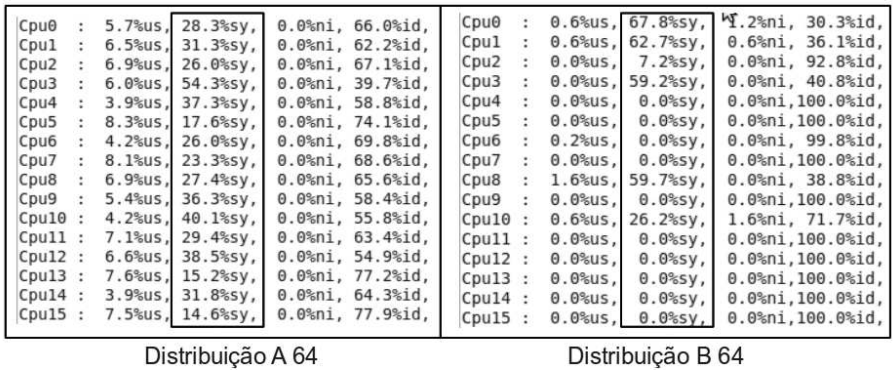
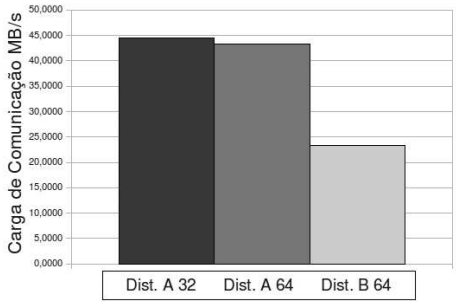


Figura 4. Desempenho de diferentes distribuições Linux - processamento

O mesmo pode ser observado no caso do teste de carga de comunicação apresentado na Figura 5 em que a distribuição A32 ficou cerca de 3% acima da distribuição A64 e a distribuição B64 ficou cerca de 47% abaixo da distribuição A32.

A Figura 6 apresenta os testes realizados para a medida de desempenho com a concorrência de *clusters* virtuais em um mesmo servidor. Foram realizados testes para 1 (utilizado como referência), 2 e 3 *clusters* concorrentes. No caso de 4 processos houve uma queda de cerca de 16% ao ser adicionado um segundo *cluster*, sendo que a perda se acentuou para 24% ao ser adicionado um terceiro *cluster*.

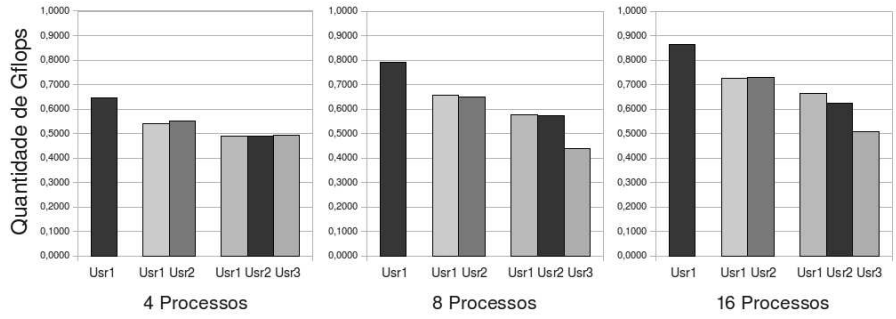
No caso de 8 processos, a perda para um *cluster* concorrente aumentou de apenas 1% em relação ao caso anterior de 4 processos (17%). Para 2 *clusters* concorrentes a perda foi de 44,5%, ou seja, quase o dobro em relação ao caso anterior. É possível notar que o desempenho do *cluster* do usuário 3 foi cerca de 24% menor em relação aos outros dois usuários. Isso se deve ao fato de o terceiro usuário ter sido inicializado por último encontrando, assim, o processamento do servidor já em uma situação de carga próxima de seu limite. Nessa situação, tanto o SO quanto o VMM não foram capazes de efetuar o balanceamento correto da carga.



	Distribuição A 32	Distribuição A 64	Distribuição B64
Média	44,4813	43,2703	23,3580
Limite Inferior	44,2852	43,0881	23,3102
Limite Superior	44,6775	43,4526	23,4058

Figura 5. Desempenho de diferentes distribuições Linux - carga de comunicação

No caso de 16 processos fica claro o efeito da concorrência sobre o desempenho dos *clusters*. Na adição de um segundo *cluster* o resultado se igualou aos anteriores, devido ao fato de haver ainda recursos no servidor capazes de absorver a carga. Porém, a adição do terceiro *cluster* acarretou na perda de cerca de 4,5% do usuário 2 em relação ao usuário 1, de 17,8% do usuário 3 em relação ao usuário 1 e de 13,3% do usuário 3 em relação ao usuário 2.



	Usuário 1		Usuário 1	Usuário 2		Usuário 1	Usuário 2	Usuário 3
4 processos	0,6449		0,5412	0,5510		0,4882	0,4908	0,4929
8 Processos	0,7920		0,6550	0,6511		0,5766	0,5750	0,4393
16 Processos	0,8652		0,7268	0,7311		0,6628	0,6238	0,5090

Figura 6. Desempenho com concorrência de usuários

Constata-se, pelos resultados obtidos, que há uma perda de desempenho, como já se esperava, do *cluster* virtual em relação ao *cluster* real, tornando-se acentuada no caso da medida de carga de comunicação, pelos motivos já citados. Essa perda pode ser reduzida pelo uso de paravirtualizadores, sob pena de redução na portabilidade. Pode ser mitigada, também, por meio do desenvolvimento e uso de bibliotecas que minimizem o efeito da perda com dispositivos de I/O, mas que ainda garantam a portabilidade em um determinado nível. No caso aqui apresentado o objetivo do projeto no qual este trabalho se insere é priorizar a portabilidade e a possibilidade de uso de um número maior de SO hospedeiros, mesmo que com um nível de desempenho menor.

Os testes tiveram como mérito a determinação do valor da perda que se observa no caso dessa escolha. A possibilidade do uso de um número maior de plataformas reais como hospedeiras, associada à coexistência de ambientes virtuais com o ambiente real,

torna-se atraente, ainda que haja perda de desempenho. Há de se enfatizar o fato de tal possibilidade ser utilizada em testes, ensino ou desenvolvimento e seus desdobramentos, sem a necessidade de dedicação exclusiva ou mudança da plataforma hospedeira.

Tais testes também mostraram a importância e o cuidado que se deve ter quando da escolha do tipo de ambiente de virtualização. Apesar de as diferenças entre paravirtualização, virtualização total e virtualização assistida por *hardware* já serem conhecidas, em função dos resultados apresentados é importante observar que outros fatores devem ser levados em consideração. A exemplo da escolha do *Kernel* (no caso das distribuições *Linux*, que são de código aberto e auditáveis), que se torna mais crítica nos casos de ambientes de virtualização proprietários. Além disso, há, no caso do *Linux*, o cuidado na escolha da distribuição pelo fato de serem utilizadas bibliotecas de 32 bits em ambientes de 64 Bits, o que pode contribuir negativamente para o desempenho [Bookman 2002].

Vale salientar que no caso das distribuições *Linux*, o *Kernel* vem sendo aprimorado de forma a melhor administrar a comunicação entre os dispositivos de entrada e saída e as máquinas virtuais. Além de um melhor gerenciamento de memória, destaca-se que, a partir da versão 2.6.32 do *Kernel* do *Linux*, foi introduzido o KSM (Kernel Samepage Merging)[REDHAT 2010]. Tal avanço consiste na varredura da memória de cada máquina virtual e, onde houver páginas idênticas de memória, estas são consolidadas em uma única página, que é então compartilhada entre todas as máquinas virtuais, o que aumenta tanto o desempenho quanto o número de máquinas virtuais que poderão ser hospedadas em uma única máquina real.

Em relação ao efeito da concorrência, foi possível notar que, enquanto havia recursos, houve balanceamento no uso dos núcleos do servidor, ainda que com certo nível de perda de desempenho. Com o aumento do número de processos - com 2 usuários, no caso de 4 processos, obteve-se 0,5412 Gflops, e, com 16 processos, 0,7268 Gflops - houve, inclusive, o aumento da capacidade de processamento. Todavia, quando a capacidade de recursos da servidora chegou ao seu limite e o balanceamento não foi possível, observou-se uma diferenciação da capacidade de processamento entre os usuários (para 3 usuários 4 processos praticamente não houve diferença, para 8 processos o usuário 3 se diferenciou e para 16 processos todos foram diferentes - Figura 6). Percebeu-se, ainda, durante os testes, o papel desempenhado pelo SO/VMM da máquina hospedeira na distribuição da carga das diversas máquinas virtuais e como isso contribuiu para o aumento do desempenho. Outro aspecto a ser considerado é a criação de um processo de escalonamento que monitorasse os recursos e, em caso de um aumento de carga de trabalho, e desde que haja recursos ociosos, rescalonasse a MV do *cluster* para tais recursos utilizando o que se conhece como migração em tempo real.

5. Conclusões

O estudo acima levantou um conjunto de questionamentos relacionados ao uso do ambiente de *Cloud* e mais especificamente sobre o custo da virtualização em HPC. Utilizamos como exemplo uma série de testes em que foram comparadas configurações e desempenho de *clusters* virtuais em relação a um *cluster* real, além de levantar alguns dos fatores que podem influenciar neste desempenho e que, por conseguinte, devem ser observados.

A escolha tanto do virtualizador como do tipo da aplicação que será executada

neste ambiente devem ser submetidos a uma análise criteriosa: itens como a infraestrutura de *hardware*, passando pela escolha do SO/VMM, inclusive com detalhamento no nível do *Kernel* e módulos, até a aplicação virtualizada propriamente dita. Tal cuidado não eliminará o custo da virtualização, mas este será minimizado.

O uso de *clusters* na forma de *appliances* apresenta grandes possibilidades, principalmente se for analisado não somente sob o aspecto do desempenho, mas também sob a ótica do desenvolvimento de testes e principalmente para o aprendizado e disseminação do uso da computação paralela na solução de problemas complexos. A capacidade de compartilhamento e a segmentação do uso dos recursos, mesmo com perda de desempenho, podem ser vistas como atraente a partir do momento em que estes recursos, por vezes ociosos, poderiam ser efetivamente utilizados. Há ainda vantagem se for levado em conta que o tempo necessário para a criação deste ambiente virtual é menor que o necessário para um ambiente real, além de apresentar um menor número de problemas relacionados a incompatibilidade em termos de *hardware*.

O estudo aqui apresentado, conquanto levante vários questionamentos pertinentes à lógica do uso de virtualização em ambientes de HPC, não esgota o assunto. Na realidade, o presente trabalho levantou, tanto durante os testes quanto ao final, uma série de questões ainda a serem respondidas. Concordamos que para o uso da virtualização em um ambiente desse nível são necessários mecanismos para a coleta de informações, sobretudo de desempenho e de carga, que não apenas permitam a transparência para seus usuários, mas que venham a fornecer subsídios para a melhoria dos serviços. Entretanto, sob o ponto de vista do seu uso para teste, ensino e desenvolvimento, em que pese a perda de desempenho, apontada nesse artigo, se mostra como uma ferramenta eficaz e que merece ser estudada, sobretudo ao poder utilizar de recursos ociosos e não dedicados.

Dentre os questionamentos levantados e que abrem novos cenários de testes, destacam-se: como implementar mecanismos de balanceamento que, em conjunto com o uso de migração em tempo real, possam minimizar o efeito da concorrência entre usuários; como os mecanismos de recuperação de falhas existentes nos virtualizadores podem auxiliar os usuários aumentando a confiabilidade na execução de suas aplicações; como se dá o relacionamento entre o uso de uma determinada aplicação em um ambiente virtual e a associação deste a um ou mais núcleos; e, finalmente, a contribuição do SO/VMM no balanceamento de carga.

O estudo até o momento realizado permite, no entanto, vislumbrar que, com o aperfeiçoamento das técnicas de virtualização, seja no nível do *hardware*, por meio da incorporação de instruções dedicadas no processador e aumento da complexidade do gerenciador de memória, seja no aperfeiçoamento do *Kernel* dos SO das máquinas hospedeiras ou por meio do desenvolvimento de aplicações dedicadas, será possível, em um futuro próximo, diminuir ainda mais as limitações existentes em termos de desempenho, permitindo difundir sua utilização em ambiente de HPC.

Referências

Bookman, C. (2002). *Linux Clustering: Building and Maintaining Linux Clusters*. Sams.

Brandt, J., Gentile, A., Mayo, J., Pebay, P., Roe, D., Thompson, D., and Wong, M. (2009). Resource monitoring and management with ovis to enable hpc in cloud computing

environments. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, Washington, DC, USA. IEEE Computer Society.

Creel, M. (2008). Pelicanhpc tutorial. UFAE and IAE Working Papers 749.08, Unitat de Fonaments de l'Analisi Economica (UAB) and Institut d'Analisi Economica (CSIC).

Dittner, R. and Rule, D. (2007). *The Best Damn Server Virtualization Book Period: Including Vmware, Xen, and Microsoft Virtual Server*. Syngress Publishing.

Evangelinos, C. and Hill, C. N. (2008). Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. Cloud Computing and Its Applications.

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*. IEEE.

Gavrilovska, A., Kumar, S., Raj, H., Schwan, K., Gupta, V., Nathuji, R., Niranjan, R., Ranadive, A., and Saraiya, P. (2007). Abstract high-performance hypervisor architectures: Virtualization in hpc systems. In *HPCVirt '07: Proceedings of the 1st Workshop on System-level Virtualization for High Performance Computing*.

Keahey, K., Foster, I., Freeman, T., and Zhang, X. (2005). Virtual workspaces: Achieving quality of service and quality of life in the grid. *Sci. Program.*, 13(4):265–275.

Kredel, H. and Merz, M. (2004). The design of the ipacs distributed software architecture. In *ISICT '04: Proceedings of the 2004 international symposium on Information and communication technologies*, pages 14–19. Trinity College Dublin.

Mergen, M. F., Uhlig, V., Krieger, O., and Xenidis, J. (2006). Virtualization for high-performance computing. *SIGOPS Oper. Syst. Rev.*, 40(2):8–11.

Nussbaum, L., Anhalt, F., Mornard, O., and Gelas, J.-P. (2009). Linux-based virtualization for HPC clusters. In *Montreal Linux Symposium*, Montreal Canada.

Ranadive, A., Kesavan, M., Gavrilovska, A., and Schwan, K. (2008). Performance implications of virtualizing multicore cluster machines. In *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, pages 1–8, New York, NY, USA. ACM.

REDHAT (2010). Kernel samepage merging. Technical report, KVM-Kernel Based Virtual Machine.

Riedel, M., Eickermann, T., Habbinga, S., Frings, W., Gibbon, P., Mallmann, D., Wolf, F., Streit, A., Lippert, T., Schiffmann, W., Ernst, A., Spurzem, R., and Nagel, W. E. (2007). Computational steering and online visualization of scientific applications on large-scale hpc systems within e-science infrastructures. In *eScience*, pages 483–490. IEEE Computer Society.

Russell, R. (2008). virtio: towards a de-facto standard for virtual i/o devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95–103.

Stanoevska-Slabeva, K., Wozniak, T., and Ristol, S. (2009). *Grid and Cloud Computing: A Business Perspective on Technology and Applications*. Springer, 1 edition.

SUNMicrosytems (2010). Virtualbox. Technical report, SUN Microsytems.

Implantando e Monitorando uma Nuvem Privada

Shirlei Aparecida de Chaves, Rafael Brundo Uriarte, Carlos Becker Westphall

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
88040 – 900 Florianópolis – SC – Brasil

{shirlei,rafael.uriarte,westphal}@inf.ufsc.br

Abstract. *This paper describes the deployment and configuration of a solution for provisioning Infrastructure as a Service or IaaS, in order to simulate common environments in day-to-day of small businesses and researchers. To this end, we used the popular open source system Eucalyptus. In addition to reporting the experiences of this deployment, we developed a solution for simple monitoring of this cloud using the software Nagios.*

Resumo. *Neste trabalho foi implantada e configurada uma solução para o fornecimento de Infraestrutura como Serviço ou IaaS (Infrascruture as a Service) visando simular ambientes comuns no dia-a-dia de pequenas empresas e pesquisadores. Para este fim, usou-se o popular e open source sistema Eucalyptus. Além de relatar as experiências desta implantação, para complementar o trabalho foi desenvolvida uma solução para o monitoramento simples desta nuvem usando o software Nagios.*

1. Introdução

O termo nuvem ou cloud é uma metáfora em relação à forma como a Internet é usualmente mostrada nos diagramas de rede - como uma nuvem. Nesses diagramas, o ícone da nuvem representa todas as tecnologias que fazem a Internet funcionar, abstraindo a infraestrutura e a complexidade que ela engloba [Velte et al. 2009]. No caso de Computação em Nuvem ou Cloud Computing, refere-se de modo geral a uma combinação de tecnologias e paradigmas (virtualização, computação utilitária, computação em grade, arquitetura orientada a serviços, entre outros) arquitetados e utilizadas de uma forma na qual se abstrai os detalhes do usuário, o qual não precisa ter todo o conhecimento necessário para manter essa infraestrutura. Utilizar serviços dessa nuvem (armazenamento, banco de dados, processamento, entre outros), é fazer computação em nuvem.

O tema computação em nuvem e as tecnologias que ele envolve é um assunto, na área de tecnologia, bastante comentado e estudado no momento, e ainda apresenta muitas questões em aberto, além de uma grande variedade de produtos e serviços oferecidos com essa finalidade. Conforme [Buyya et al. 2008], o primeiro pico de popularidade do termo em ferramentas de busca na web como o Google, foi em final de 2007, voltando a ganhar força no terceiro bimestre de 2008. Toda essa curiosidade em relação ao tema computação em nuvem, além da disponibilidade de diversas ferramentas e serviços que são apresentados sob esse guarda-chuva, alguns pagos como o EC2 da Amazon, levou ao interesse em se conhecer qual a viabilidade de se construir um ambiente de computação em nuvem e monitorá-lo através de ferramentas open source, aproveitando-se o hardware

disponível, mesmo que não seja projetado para tal finalidade, especialmente na categoria Infraestrutura como Serviço ou IaaS (Infrastructure as a Service), a qual é responsável por entregar uma completa infraestrutura computacional. Tipicamente, nesta forma de entrega, são fornecidas máquinas virtuais para o usuário final, que as usa conforme as suas demandas. A infraestrutura está no datacenter do provedor ou espalhada na nuvem e o usuário não tem detalhes sobre a origem de seu fornecimento.

Não se tem no momento um estudo referência no assunto, porém os autores acreditam que há um grande parque tecnológico instalado em pequenas empresas e laboratórios acadêmicos com grande potencial de aproveitamento, o qual pode ser gerenciado e aproveitado como uma nuvem privada e eventualmente até expandido para se tornar uma nuvem híbrida, por exemplo. Esse parque tecnológico pode ser composto especialmente de computadores com espaço de armazenamento de sobra, processamento e memória subutilizados, recursos de rede, entre outros. Essa subutilização pode ser exemplificada na situação de desktops e notebooks utilizados apenas para editoração simples de textos e outras rotinas administrativas que não requerem grande processamento, por exemplo, e até mesmo em algumas situações na questão de turnos de trabalho em que não há expediente na organização e esse equipamento fica todo ou parcialmente ocioso.

A principal motivação deste artigo então é, através de um estudo de caso, demonstrar a viabilidade de implantação e configuração de uma nuvem privada, utilizando-se apenas software open source, como o Eucalyptus, um dos populares softwares de infraestrutura para computação em nuvem, a ferramenta de monitoramento Nagios, entre outros. Além dessa viabilidade, outro interesse foi o levantamento dos passos necessários no projeto e implantação de uma nuvem privada, especialmente num ambiente de menor escala.

O artigo está organizado da seguinte forma. Na Seção 2, apresentam-se os trabalhos relacionados. Na Seção 3, é feita uma descrição do ambiente de testes e das ferramentas utilizadas. Na Seção 4, é apresentado o Estudo de Caso e os testes que foram efetuados. Na Seção 5, são apresentados os resultados obtidos. Na Seção 6, são apresentadas as considerações finais e trabalhos futuros.

2. Contextualização e Trabalhos Relacionados

Muitos foram os trabalhos de pesquisa e desenvolvimento que possibilitaram o desenvolvimento do que hoje se convencionou chamar computação em nuvem. Entre eles podemos citar a computação utilitária (do inglês *utility computing*), a computação em grade (*grid computing*), virtualização, computação em cluster e computação distribuída de modo geral. Diversos são os autores que sustentam essa constatação, entre eles podemos citar [Foster et al. 2008], [Vouk 2008], [Lim et al. 2009]. Com essa definição em mente, diversos são os produtos e serviços que são apresentados sob esse termo, o qual engloba diversas categorias, especialmente relacionadas ao modo que os serviços são entregues, como, Software como Serviço ou SaaS (Software as a Service), Plataforma como Serviço ou PaaS (*Platform as a Service*), Infraestrutura como Serviço, entre outras, dependendo do autor.

Além dessa categorização com relação aos serviços oferecidos, também há uma categorização com relação à abrangência e acesso à nuvem. A classificação geralmente citada é a de nuvem pública (public cloud), nuvem privada (private cloud) e nuvem híbrida (hybrid cloud).

De acordo com [Armbrust et al. 2009], o software e hardware de um datacenter é a nuvem e se essa nuvem for disponibilizada para o público em geral no modo pay-as-you-go, ela é chamada nuvem pública. Por outro lado, o datacenter interno à uma organização, não disponível ao público em geral, é a nuvem privada. Segundo [Sotomayor et al. 2009], o objetivo principal de uma nuvem privada não é vender capacidade pela Internet, através de interfaces acessíveis ao público em geral, mas sim dar aos usuários locais uma infraestrutura ágil e flexível para suportar workloads de serviços dentro de seu próprio domínio administrativo.

Uma nuvem privada, no entanto, pode dar suporte à uma nuvem híbrida, através da complementação da capacidade da infraestrutura local com a capacidade computacional de uma nuvem pública [Sotomayor et al. 2009]. A nuvem privada/híbrida também pode permitir acesso remoto através de interfaces remotas, como a de Web Services que a Amazon EC2 utiliza [Sotomayor et al. 2009].

Apesar de ser um conceito relativamente novo, o tema tem sido bastante estudado e pode-se citar vários trabalhos relacionados à migracão de serviços ou processos da organização para as nuvens. Esses trabalhos normalmente envolvem passos que devem ser tomados e preocupações que devem ser levadas em consideração na análise dos potenciais provedores, mas geralmente giram em torno da migração para uma nuvem pública. Por exemplo, em [CSAGuide 2009], são organizados em domínios os principais focos com relação à segurança e então trabalhados na forma de um guia, com instruções e passos que podem ser postos em prática. Em [Linthicum 2009], é apresentado um processo com passos chaves a serem seguidos para encontrar a categoria ou categorias de nuvem mais adequada, além de possíveis candidados a provedores para os quais a empresa poderia migrar seus processos. Até a escrita desse artigo, no entanto, os autores não localizaram material técnico ou científico a respeito da criação e implantação de uma nuvem privada em pequenas empresas e ambientes domésticos. Segundo [Khalid 2010] existem vários problemas na migração total do ambiente local para uma nuvem pública, sendo segurança e privacidade os principais exemplos. O mesmo autor também sugere que o ideal é começar com uma nuvem privada e depois integrar esta nuvem em uma nuvem pública, abordagem que os autores do presente artigo acreditam ser importante especialmente quando se trata de pequenas empresas.

3. Descrição do Ambiente e Ferramentas Utilizadas

A implantação do ambiente de testes foi feita utilizando-se as ferramentas de hardware e software na Tabela 1.

Conforme citado anteriormente, o principal interesse desse trabalho é verificar a viabilidade de se construir um ambiente de computação em nuvem utilizando-se software open source e especialmente hardware que chamaremos de legado, não tendo sido projetado para computação em nuvem, mas com recursos suficientes para suportar a instanciação e execução de máquinas virtuais. Esse foi o principal princípio norteador na escolha dos software utilizados.

Software para Computação em Nuvem

O Eucalyptus foi escolhido por ser uma plataforma para computação em nuvem open source, bastante documentada e oferecer suporte para diversas distribuições linux.

Tabela 1. Hardware e Software Utilizados no Estudo de Caso

Hardware	Software	Papel no Ambiente
Notebook Acer Aspire 5920, Processador: Intel® Core™ 2 Duo T5750, Cache do Processador: 2MB L2 Cache, Memória RAM: 4GB DDR2 667MHz, Disco Rígido (HD): 250GB SATA 5400rpm (o sistema operacional utilizado foi instalado em uma partição de 90 GB)	OpenSUSE 11.1 64 bits, Eucalyptus release 1.6.1, Xen versão 3.3.1, Nagios versão 3.0.6	Controlador de cluster, Controlador de nuvem, Controlador de armazenamento, Nodo
CPU HP Pavilion 6205BR, Processador: Pentium 4 3.6GHz, Memória RAM: 1 GB DDR2 333 MHz, Disco Rígido (HD): 250GB (o sistema operacional foi instalado em uma partição de 80 GB)	OpenSUSE 11.1 32 bits, Eucalyptus release 1.6.1, Xen versão 3.3.1	Nodo
Roteador Wireless Intelbras WRG 240 E 54Mbps		Servidor DHCP

Além disso, por oferecer uma interface compatível com a EC2 da Amazon, amplamente utilizada e considerada por alguns um padrão de facto quando se trata de serviços para a nuvem. Por fim o seu design hierarquizado visa e facilita (mas não limita) o seu uso em laboratórios e em pequenas e médias empresas [Nurmi et al. 2009].

Hipervisor

A escolha do hipervisor de modo geral está atrelada ao software para computação em nuvem escolhido. No caso do Eucalyptus, na versão open source, há suporte para dois hipervisores também open source: o Xen e o KVM. O hipervisor ou monitor de máquina virtual é quem permite que seja executado o sistema operacional hóspede, construindo as interfaces virtuais para as interfaces reais do sistema hospedeiro. Há diversas maneiras de se implementar o mecanismo de virtualização e cada hipervisor pode estar apto a trabalhar com uma dessas técnicas e eventualmente mais de uma. Para o caso dos dois hipervisores citados, Xen é o que se adequa ao estudo de caso, pelo fato de implementar o mecanismo de paravirtualização, técnica na qual uma camada de hardware virtual muito similar ao hardware real é adicionada ao hipervisor [Santos and Charão 2008], não precisando de suporte nativo à virtualização pelo hardware. Para acomodar o uso dessa técnica, o kernel do sistema operacional precisa ser adaptado. No caso do hipervisor KVM (*Kernel-based Virtual Machine*), o hardware utilizado deve possuir suporte à virtualização pelo processador, característica encontrada apenas em computadores de grande porte, como mainframes, até o lançamento das extensões IVT (*Intel Virtualization Technology*) pela Intel e da AMD-V (AMD *Virtualization*), pela AMD [Santos and Charão 2008]. O próprio KVM é recente, tendo sido lançado oficialmente em 2007. Essa virtualização por suporte

nativo no hardware é chamada de Máquina Virtual de Hardware ou *Hardware Virtual Machine* (HVM). No caso do KVM, não há a necessidade de se alterar o kernel do sistema operacional, pois ele está disponível como um módulo no kernel padrão na maioria das distribuições linux.

No hardware utilizado para testes, ambos os processadores eram Intel, mas nenhum deles com suporte à virtualização, portanto o hipervisor escolhido foi o Xen. É importante salientar, no entanto, que há diversos estudos comparando performance de diversos hipervisores e cada um tem suas vantagens em um ou outro aspecto estudado ou testado. A escolha entre um e outro, em geral, está relacionada à flexibilidade de uso, performance e considerações estratégicas/estudo e planejamento do ambiente onde vai ser utilizado [Cerbelaud et al. 2009].

Sistema Operacional das Máquinas Fisicas

O sistema operacional OpenSUSE foi escolhido por ter suporte ao Eucalyptus e também suporte nativo ao Xen, sendo que esse último tem sua instalação e configuração bastante facilitadas pela ferramenta de instalação e configuração fornecida pelo openSUSE, o YaST.

Monitoramento

O Nagios é uma ferramenta popular e open source para o monitoramento de redes que vem sendo desenvolvida a mais de 10 anos. Por isso conta com ampla documentação e suporte da comunidade de código aberto. Além da sua flexibilidade de uso (que permite a criação de módulos específicos através de plugins, por exemplo) e por ser open source, considerou-se na sua escolha o fato deste software estar sendo usado pela plataforma Eucalyptus. Até o momento do presente trabalho, o Eucalyptus fornece um script, mesmo que limitado inicialmente, para geração das configurações básicas para o Nagios monitorar os principais componentes do Eucalyptus (ver Seção 3.1).

3.1. Eucalyptus

O Eucalyptus é composto por quatro componentes de alto nível, cada um com sua própria interface de web service. São eles [Nurmi et al. 2009]:

- Controlador de nodo (*node controller – nc*): controla a execução, inspeção e término das instâncias de VMs, no host onde está sendo executado.
- Controlador de cluster (*cluster controller – cc*): faz o agendamento e coleta informações da execução das VMs num controlador de nodo específico e gerencia instâncias de rede virtual.
- Controlador de armazenamento (*storage controller – walrus*): serviço de armazenamento baseado em put/get que implementa a interface S3 (*Simple Storage Service*) da Amazon, provendo um mecanismo para acesso e armazenamento de imagens de VMs e dados de usuários.
- Controlador de nuvem (*cloud controler*): ponto de entrada na nuvem para usuários e administradores. Ele indaga os gerenciadores de nodos sobre informações acerca de recursos, faz decisões de agendamento de alto nível e as implementa através de requisições aos cluster controllers.

Para a configuração de rede, há quatro modos disponíveis, conforme listado abaixo [Eucalyptus 2009]:

a) *System*: modo mais simples, com menos recursos de rede. Nesse caso, não é o Eucalyptus que fica responsável pela atribuição de IPs aos nodos, mas um servidor DHCP externo ao Eucalyptus. O Eucalyptus designa um MAC aleatório para a VM antes do boot e associa a interface ethernet da máquina virtual à interface física através da ponte (bridge) Xen que foi configurada para o nodo local. O endereço IP então é atribuído à essa máquina virtual pelo servidor DHCP da mesma forma que é atribuído a qualquer outra máquina física dessa rede.

b) *Static*: nesse modo, o administrador configura uma lista de pares MAC/IP, que é utilizada pelo Eucalyptus para atribuir às VMs. A parte de associar a interface de rede da máquina virtual à interface física é similar ao modo anterior. A vantagem desse modo seria um maior controle sobre os IPs que se quer associar às VMs.

c) *Managed*: esse é o modo mais completo, oferecendo a possibilidade de se isolar as máquinas virtuais em uma rede de máquinas virtuais, de se criar grupos seguros (security groups), com regras de ingresso, assim como definir uma lista de IPs públicos, que os usuários podem utilizar para atribuir às suas VMs durante o boot ou em tempo de execução.

d) *Managed-NOVLAN*: mesmas características do modo gerenciado, mas sem a possibilidade de se isolar a rede de VMs.

A instalação do cloud controller, do clust controller e do node controller são triviais, a configuração do hipervisor e especialmente da rede é que podem requerer maior atenção, conforme o modo de rede utilizado e o ambiente em que se está configurando.

Características particulares de configuração do Eucalyptus são feitas no arquivo eucalyptus.conf. Por exemplo, pode-se definir o número de CPUs ou núcleos (*core*) a serem utilizados no nodo. Na configuração padrão, se utiliza o máximo de CPUs ou núcleos que encontrar para instanciar as máquinas (no caso do ambiente de testes, como os dois nodos utilizados eram Core 2 duo, seria por padrão possível instanciar duas máquinas virtuais em cada um). Para testes, essa variável foi configurada para MAX CORES = 4. Ao contrário do que acontece com CPUs ou núcleos, não é possível compartilhar memória entre instâncias, isto é, não é possível utilizar mais memória do que a memória física disponível, mesmo alterando a configuração da variável MAX MEM, na versão utilizada do Eucalyptus (versão 1.6.1).

4. Estudo de Caso

Para a realização dos experimentos, foi projetado um estudo de caso levando-se em consideração uma nuvem privada, conforme conceito de nuvem privada já discutido na Seção 2. Essa categoria se mostra atrativa por aproveitar recursos da organização e manter o perímetro de segurança, uma vez que a segurança em computação em nuvem é um tema com muitos questionamentos em aberto e provavelmente, se não o primeiro, um dos grandes motivos para que as empresas invistam em sua própria estrutura de computação em nuvem. O gerenciamento dessa nuvem passa também a ser interno à organização, que, no mínimo, poderá obter uma melhor compreensão sobre o que se trata e atingir um maior nível de maturidade no tema se resolver futuramente migrar para uma nuvem pública ou eventualmente, transformar a sua nuvem privada em uma nuvem híbrida.

O ambiente utilizado nos testes foi construído utilizando-se os recursos de hard-

ware e software descritos na Seção 3. A configuração de rede utilizada para o Eucalyptus foi no modo system, no qual, conforme descrito na Seção 3.1, os IPs são atribuídos pelo servidor DHCP existente na rede da mesma forma que são atribuídos às máquinas físicas, não havendo um controle através de uma lista específica de IPs como no modo static. As imagens de máquinas virtuais utilizadas foram as imagens pré-configuradas disponibilizadas no site do Eucalyptus. No momento do download, estavam disponíveis as seguintes imagens, tanto para o KVM quanto para o Xen:

- Ubuntu 9.04 (32 bits e 64 bits)
- Debian 5.0 (32 bits e 64 bits)
- CentOS 5.3 (32 bits e 64 bits)
- Fedora 10 (32 bits e 64 bits)

As imagens citadas não fornecem suporte ao Nagios (plugins ou arquivos prévios de configuração). Elas foram então adaptadas, de modo a oferecem esse suporte. O plugin NRPE (*Nagios Remote Plugin Executor*) foi instalado e configurado para o monitoramento dos seguintes serviços:

- Numero de usuários logados no sistema;
- Processamento das CPUs;
- Memória utilizada/disponível nos discos locais;
- Número total de processos ativos na máquina.

4.1. Monitoramento

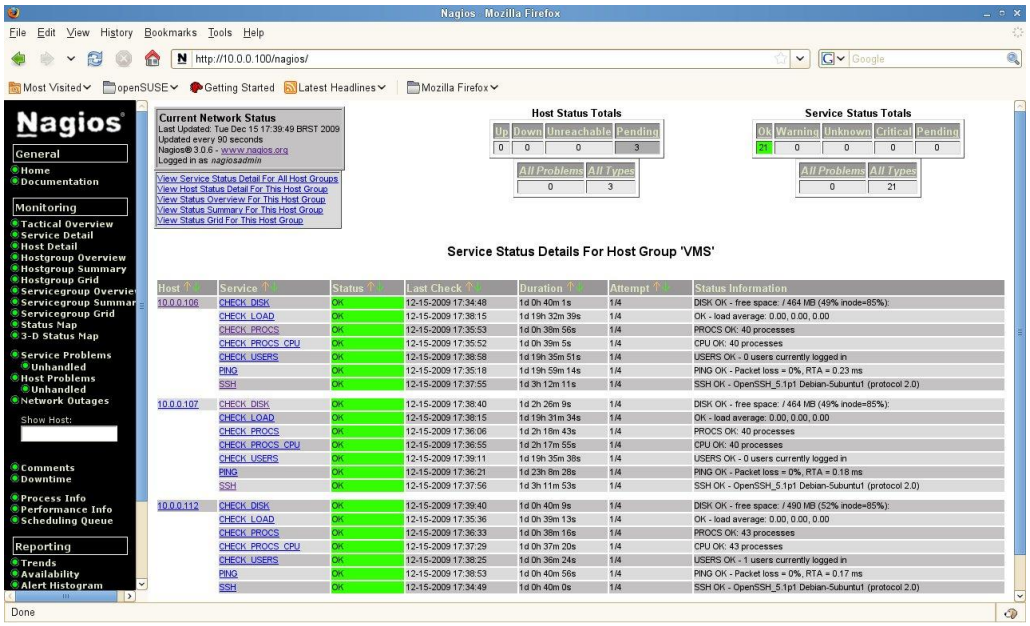


Figura 1. Tela de detalhamento dos serviços monitorados pelo Nagios para as VMs

Como em todo caso de um sistema distribuído, chega-se à questão de que se o monitoramento será centralizado ou distribuído, especialmente devido a questões de escalabilidade. Nesse trabalho optou-se pela abordagem centralizada por reduzir a complexidade da estrutura e, como se queria demonstrar a viabilidade de configuração de uma nuvem privada em ambientes de menor escala, não se considerou a escalabilidade um

fator essencial no sistema de monitoramento. O Nagios foi instalado na mesma máquina em que foi instalado o *cloud controller*. A partir da versão 1.6.1, o Eucalyptus provê um script inicial para monitoramento passivo básico (*load* e *ping*) dos nodos e máquina(s) onde estão instalados o *cloud controller*, *storage controller*, *cluster controller*, *node controllers* e nenhuma funcionalidade para monitoramento das máquinas virtuais. Um script na cron, executado em um intervalo de tempo configurável, efetua a coleta de informações dos nodos.

Esse script inicial fornecido pelo Eucalyptus foi utilizado como recurso base no sentido de se adicionar capacidades de monitoramento das máquinas virtuais, para expandi-lo e utilizá-lo como um framework para monitoramento. Os algoritmos foram formulados para serem genéricos sempre que possível e utilizam comandos de interação com o Eucalyptus fornecidos pela ferramenta *euca2ools*, conforme pode ser visualizado no trecho de código da Listagem 1, o qual obtém a lista de IPs das instâncias de máquinas virtuais sendo executadas. O euca2ools é uma ferramenta de linha de comando para interação com web services que exportam uma API (*Application Programming Interface* ou Interface de Programação de Aplicativos) baseada em REST/QUERY, compatível com os serviços EC2 e S3 da Amazon (ou seja, essa mesma ferramenta pode ser utilizada para os serviços citados da Amazon, desde que se tenha uma conta com esses serviços).

```
1  get_vms(){
2          LIST=`euca-describe-instances | awk '/([1-9]|[1-9]\d|1\d{2}|2[0-4]\d|25[0-5])$/ {print $4}'`
3          #removing new line character and putting into an array
4          for s in $(echo "$LIST" | tr '\n' ' ')
5          do
6                  VMS_LIST=( "${VMS_LIST[@]}" "$s" )
7          done
8  }
```

Listagem 1: Função codificada para obter a lista de IPs atribuídos às VMs.

```
linux-sqrq:/home/shirlei # euca-describe-instances
RESERVATION        r-495F089A        admin        default
INSTANCE        i-39BB0673        emi-58131665        10.0.0.112        running        fookey        0
m1.small        2009-12-14T18:47:43.52Z        cluster_test        eki-942D1709        eri-8E9716F6
RESERVATION        r-461D074F        admin        default
INSTANCE        i-453D0799        emi-57FB1670        10.0.0.107        running        fookey        0
m1.small        2009-12-13T20:28:12.825Z        cluster_test        eki-942D1709        eri-8E9716F6
RESERVATION        r-394806C9        admin        default
INSTANCE        i-3ADA06FD        emi-57FB1670        10.0.0.106        running        fookey        0
m1.small        2009-12-13T20:17:27.369Z        cluster_test        eki-942D1709        eri-8E9716F6
```

Listagem 2: Resultados obtidos com o comando euca-describe-instances

```
linux-sqrq:/home/shirlei # xm list
Name        ID   Mem    VCPUs    State   Time(s)
Domain-0    0    3497     2      r-----  136818.1
i-39BB0673  6     128     1      -b----      41.1
i-3ADA06FD  1     128     1      -b----      68
i-453D0799  2     128     1      -b----      72
```

Listagem 3: Resultados obtidos com o comando xm list

No código mostrado na Listagem 2, usou-se diretamente o comando euca-describe-instances pela facilidade de se obter diretamente os IPs que foram associados às instâncias de máquinas virtuais, endereços esses essenciais no monitoramento a ser efetuado. Se fosse usado o comando xm list (comando do Xen para listar as instâncias em execução) ou, mais genericamente ainda, virsh list (comando que utiliza a biblioteca libvirt e pode ser usado para qualquer hipervisor suportado por essa biblioteca), os IPs teriam que ser obtidos de outra forma. Nas Listagens 2, 3 e 4 são mostrados os resultados obtidos com os três comandos citados. Percebe-se na Listagem 2 que o resultado obtido

```
linux-sqrq:/home/shirlei # virsh list
Id_____Name_____State_
0        Domain-0        running
1        i-3ADA06FD        idle
2        i-453D0799        idle
6        i-39BB0673        idle
```

Listagem 4: Resultados obtidos com o comando virsh list

com o comando euca-describe-instances nos fornece diretamente o IP atribuído a cada instância.

Outra diferença em relação ao monitoramento básico fornecido pelo Eucalyptus para os seus componentes, foi em relação à abordagem ativa/passiva no monitoramento. O tipo de monitoramento utilizado foi o ativo, ou seja, o Nagios é quem inicia a checagem das métricas, com o auxílio do plugin para monitoramento remoto NRPE, adicionado às imagens utilizadas.Conforme listado na seção anterior, considerou-se as seguintes métricas ou serviços, já pré-configuradas na imagem a ser carregada nas VMs:

Monitoramento através do NRPE:

- *Load* (carga)
- Disco
- Usuários logados
- Utilização de cpu por processos
- Total de processos em execução

As métricas acima foram escolhidas por serem as métricas mais comuns e genéricas, já que no momento atual da pesquisa não se está testando nenhum serviço ou carga de trabalho que pudesse afetar ou ser afetada por essas métricas. O foco principal é avaliar a monitorabilidade do ambiente configurado.

Monitoramento através da checagem direta pelo Nagios:

- SSH

Após o instaciamento da VM pelo Eucalyptus, o acesso à mesma é feito via ssh com o uso de chave gerada para o usuário que solicitou o instanciamento. Se por algum motivo a conexão ssh não for possível, o acesso à máquina para a instalação de novos programas ou outra atividade administrativa não pode ser feito e, conforme o caso, a VM pode se tornar inútil. Portanto, o monitoramento desse serviço pode ser essencial para uma política de término de VMs, por exemplo.

- Ping

O teste de ping tem como objetivo testar a alcançabilidade das VMs, uma vez que elas fazem uso de serviços que utilizam do protocolo IP, como o SSH. Eventualmente, essa métrica pode não ser útil se houver uma política para que as VMs não respondam à essas requisições para evitar tráfego na rede. Porém, não é o caso do ambiente configurado para o estudo de caso.

Feitas as alterações necessárias nas imagens para VMs utilizadas e o script para coleta dos IPs das VMs e geração do arquivo de configuração para o Nagios, o processo de monitoração em si é trivial, isto é, basta adicionar o arquivo vms.cfg, com as definições dos serviços a serem monitorados gerado com o script cujo trecho de código é mostrado acima, no diretório apropriado e reiniciar o serviço de monitoramento do Nagios. Na Figura 1 é mostrada a tela com as informações de monitoramento gerada pelo Nagios, para o *hostgroup* VMS (*hostgroup* gerado para conter as máquinas virtuais).

5. Resultados Obtidos

Como resultados cita-se a extensão do monitoramento básico pelo Nagios oferecido pelo Eucalyptus, na forma de adição de funcionalidade que agrega também as máquinas virtuais ao monitoramento. Para facilitar a visualização do que é máquina física e do que é máquina virtual, na interface de gerenciamento do Nagios, as VMs foram agrupadas em um *hostgroup* próprio, conforme pôde ser visto na Figura 1.

Também pode ser citado com um resultado derivativo o fato de se mostrar a facilidade de adaptação das imagens para máquinas virtuais pré-construídas disponibilizadas pelo Eucalyptus ou outra entidade, o que retira do administrador a necessidade de se gerar essas imagens do zero, mas também lhe permite adaptá-las para as necessidades específicas de seu domínio administrativo.

Com a experiência relatada se pôde visualizar e ponderar os passos que são necessários na construção desse ambiente. De modo geral, são os passos necessários na montagem de diversos ambientes computacionais, porém o propósito de cada um, além do proposito geral de planejamento, tem algumas particularidades, conforme relata-se a seguir.

1. *Levantamento e análise do hardware disponível;*
   O hardware disponível influencia em escolhas como, por exemplo, qual hipervisor utilizar.
2. *Investigação dos softwares para computação em nuvem existentes e adequados ao hardware disponível;*
   Muitos produtos são vendidos sob o termo computação em nuvem, assim como muitos produtos *open source* estão disponíveis para testes, portanto é necessária uma investigação para diferenciar os produtos e verificar qual ou quais seriam mais adequados para o interesse da organização, além de serem adequados ao hardware disponível.
3. *Configuração de um ambiente inicial para testes, o qual poderia servir de protótipo para usos diversos e para que a organização se familiarizasse com o uso.*
   Essa fase é a de aplicação prática do levantamento feito nos passos anteriores e é também a fase que vai requerer maior disponibilidade técnica de pessoal.

Além dos aspectos técnicos, pode-se citar também que, de certa forma, contribuiu-se para a desmistificação do tema computação em nuvem para ambientes de menor escala, como uma pequena empresa ou laboratório acadêmico. Muito se fala e se pesquisa sobre o assunto, mas para ambientes de pequeno porte existe pouca abordagem. Com esse artigo mostra-se a viabilidade da aplicação do conceito e suas vantagens.

6. Considerações Finais e Trabalhos Futuros

Ao se analisar cuidadosamente e se configurar um ambiente de testes que se enquadre na definição de nuvem privada, com as ferramentas para computação em nuvem disponíveis, é possível se ter um ambiente de computação em nuvem sem maiores investimentos a não ser eventualmente o de pessoal (configuração do ambiente e suporte). A atratividade criada em torno da nuvem privada, além dos benefícios em relação à segurança, se dá também em relação a um quesito bastante em voga na atualidade: o consumo consciente.

Ao se utilizar melhor os recursos computacionais já existentes no data center da empresa, pode-se reduzir o consumo de energia, postergar a aquisição de novos equipamentos e em última instância, reduzir o volume de lixo eletrônico, que causa contaminação e outros problemas ambientais e sociais em diversos lugares do mundo.[1]

Plataformas de software para computação em nuvem como o Eucalyptus, por exemplo, não exigem hardware dedicado e auxiliam no gerenciamento e integração de diversos recursos computacionais, desde um desktop a um moderno servidor. Essa característica fica bastante evidente no ambiente de testes utilizados, o qual foi um ambiente que se pode chamar de 'caseiro', utilizando-se desktop e notebook de usos domésticos.

Como próximos passos de pesquisa, se pretende analisar os possíveis impactos dessas modificações na infraestrutura, tanto no quesito desempenho, como também no quesito gerenciamento de atualizações. Essa análise possibilitará se trabalhar melhor o conceito de monitoramento passivo/ativo, no sentido de identificar qual seria a melhor solução para cenários similares, além de se definir melhor questões como intervalo de monitoramento, adição dinâmica de novos recursos (VMs instanciadas) ao Nagios ou outra ferramenta que venha a ser utilizada. O ambiente de testes será expandido para se tornar um *testbed* para avaliações e testes diversos, como desempenho de aplicações, impacto de compartilhamento de recursos, entre outros. Para uma melhor configuração desse ambiente de testes, se aprofundará o estudo das necessidades de pequenas/médias empresas, além de se buscar o desenvolvimento um framework para o monitoramento e gerenciamento de computacão em nuvens voltada para as mesmas.

Pretende-se ainda aplicar mecanismos e políticas de segurança neste contexto, dando continuidade ao trabalho relacionado a Acordo de Nível de Serviço relacionado à Segurança em computação em nuvem, conforme em [Chaves et al. 2010].

Referências

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Buyya, R., Yeo, C. S., and Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 5–13, Washington, DC, USA. IEEE Computer Society.

Cerbelaud, D., Garg, S., and Huylebroeck, J. (2009). Opening the clouds: qualitative overview of the state-of-the-art open source vm-based cloud management platforms. In *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, pages 1–8, New York, NY, USA. Springer-Verlag New York, Inc.

[1] Ver o caso de Guiyu, na China, considerada a capital mundial do e-wast (literalmente, lixo eletrônico)

Chaves, S. A., Westphall, C. B., and Lamin, F. R. (2010). Sla perspective in security management for cloud computing. In *The Sixth International Conference on Networking and Services - ICNS 2010*, Cancun, Mexico.

CSAGuide (2009). Security guidance for critical areas of focus in cloud computing. Technical report, Cloud Security Alliance, http://www.cloudsecurityalliance.org/guidance/csaguide.v2.1.pdf.

Eucalyptus (2009). Eucalyptus documentation. `http://open.eucalyptus.com`.

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10.

Khalid, A. (2010). Cloud computing: Applying issues in small business. *Signal Acquisition and Processing, International Conference on*, 0:278–281.

Lim, H. C., Babu, S., Chase, J. S., and Parekh, S. S. (2009). Automated control in cloud computing: challenges and opportunities. In *ACDC '09: Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 13–18, New York, NY, USA. ACM.

Linthicum, D. (2009). Selecting the right cloud. Technical report, InfoWorld, http://www.infoworld.com/cloud-deepdive.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The Eucalyptus open-source cloud-computing system. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, volume 0, pages 124–131, Washington, DC, USA. IEEE.

Santos, R. C. M. d. and Charão, A. S. (2008). Análise comparativa de desempenho do hipervisor xen: Paravirtualização versus virtualização total. *WSCAD 2008: IX Workshop em Sistemas Computacionais de Alto Desempenho*.

Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. (2009). Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22.

Velte, T., Velte, A., and Elsenpeter, R. (2009). *Cloud Computing, A Practical Approach*, chapter 1, page 3. McGraw-Hill Osborne Media, first edition.

Vouk, M. A. (2008). Cloud computing: Issues, research and implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 31–40.

Uni4Cloud – Uma Abordagem para Implantação de Aplicações sobre Múltiplas Nuvens de Infra-Estrutura

Américo Sampaio, Matheus Cunha, Nabor Mendonça, Marcelo Barros, Thiago Leite, Roberto Costa, Michel Vasconcelos

Mestrado em Informática Aplicada (MIA) – Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321, Bloco J, Sala 30, CEP 60811-905 Fortaleza, CE, Brazil

{americo.sampaio, mathcunha, nabor, marcelobarrosmia, robertocostalima, michelav}@unifor.br, thiago.carvalho@serpro.gov.br

Abstract. *Cloud Computing is a computing paradigm that aims at providing on demand access, over the Internet, to a pool of resources (e.g., virtual machines, storage, networks and applications) that can be rapidly provisioned. Clouds offer high scalability and flexibility for using these computing resources which are normally provisioned as services (offered via an API) and charged as utilities such as electricity where users pay for the resources they actually consume (e.g., CPU-Hours used and data stored). This paper presents an approach that allows to model, deploy and configure complex web applications in multiple infrastructure clouds.*

Resumo. *A computação em nuvem é um modelo computacional que permite o acesso, sob demanda e através da Internet, a um "pool" de recursos computacionais (redes, servidores virtuais ou físicos, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados. As nuvens oferecem alta escalabilidade e flexibilidade de uso destes recursos, acopladas a um novo modelo de negócio onde os usuários dessas plataformas pagam apenas pelos recursos (por exemplo, tempo de processamento e volume de dados armazenados) que de fato consumirem (modelo de pagar pelo uso como serviços de Energia Elétrica). Este artigo apresenta uma abordagem que facilita a modelagem, implantação, configuração e distribuição de aplicações web complexas em múltiplas nuvens que oferecem servidores virtuais como recurso, chamadas de nuvens de infra-estrutura (IaaS).*

1. Introdução

A computação em nuvem é um novo paradigma de desenvolvimento, distribuição e implantação de aplicações na Internet que tem recentemente atraído bastante interesse da indústria de software e da comunidade acadêmica [Armbrust et al. (2009), Leavitt 2009, Galán et al. 2009]. Empresas de consultoria renomadas como Merryl Lynch [Lynch 2008] prevêem que o mercado de computação em nuvem terá um potencial de aplicação de 160 bilhões de dólares em 2011. A Gartner [Pring and Desisto 2008, Smith and Austin 2009] por sua vez estima que em 2011 75% dos serviços de software (SaaS) utilizarão algum tipo de infra-estrutura de nuvem. Já a empresa de marketing IDC aponta que o gasto diretamente ligado a investimentos em soluções de computação em

nuvem deve crescer de 16 bilhões de dólares em 2008 para 42 bilhões de dólares em 2012 [Gens 2008].

O interesse da indústria em adotar a computação em nuvem se dá principalmente pelo fato deste modelo permitir utilizar os recursos computacionais de forma mais econômica e otimizada, possibilitando uma redução de custos operacionais (menos máquinas, manutenção simplificada, redução de gastos com energia e resfriamento) [Armbrust et al. (2009)], o que representa um fator extremamente relevante considerando-se o atual cenário de crise econômica mundial e redução de gastos com tecnologia da informação (TI). Esta economia ocorre devido ao fato que empresas clientes das nuvens não precisam investir para adquirir (comprando hardware, coollers, alugando espaço físico) e administrar (contratando pessoal) sua própria infra-estrutura. Ao invés disso, estas empresas podem optar por alugar a infra-estrutura necessária de outras empresas que possuem enormes *data centers* (como Google e Amazon) e, com isso, conseguem atingir economias de escala oferecendo tais serviços a um baixo custo a qualquer cliente. Um exemplo ilustrativo de como este modelo pode ser benéfico é descrito em [Derek 2007]. Este exemplo mostra que o departamento de TI do jornal New York times conseguiu cumprir (usando um software de processamento em alto desempenho baseado no Hadoop executado sobre a plataforma de nuvem da Amazon EC2 [EC2 2009]) uma tarefa de converter parte do seu acervo de reportagens digitalizadas em imagens em formato escaneado ".tiff" para formato PDF, para que fosse integrado e disponibilizado no seu web site. A tarefa foi cumprida em menos de 24 horas e custou milhares de dólares a menos (e precisou de muito menos tempo) do que custaria se o desenvolvimento fosse feito internamente no seu departamento de TI. Caso a equipe de TI do NY times não tivesse usado a plataforma EC2 seria necessário adquirir uma grande quantidade de servidores que depois ficariam ociosos.

Este novo paradigma vem também atraindo atenção de diversos pesquisadores que apontam a computação em nuvem como uma relevante e desafiante área de pesquisa que evolui conceitos já consolidados como virtualização, computação em grade e computação utilitária. Além disto, pesquisadores apontam que a computação em nuvem traz benefícios como escalabilidade de aplicações complexas, maior disponibilidade e tolerância a falhas devido à alta capacidade da infra-estrutura e da sua adaptação dinâmica e sob demanda.

Um dos grandes desafios [Galán et al. 2009, VMware 2009] do atual cenário de computação em nuvem é possibilitar um mecanismo para implantar aplicações complexas que podem ser compostas de vários componentes (e.x. banco de dados, servidor *web*, servidor de aplicação, balanceador de carga) e serem "empacotadas" em uma ou várias máquinas virtuais que podem utilizar infra-estruturas de nuvens distintas de forma fácil e automatizada. Normalmente cada nuvem oferece uma forma proprietária de "empacotar" uma aplicação além de oferecer APIs também proprietárias para realizar a implantação da aplicação.

Este artigo apresenta a abordagem Uni4Cloud que visa facilitar a implantação de aplicações em múltiplas nuvens através da criação de ferramentas e componentes de software implementados com base em padrões abertos e soluções "open source" existentes. Dentre as principais características da abordagem, destacam-se:

- Permite criar um modelo que contém informações sobre os componentes de software que fazem parte de uma aplicação (ex: servidor *web*, servidor de aplicação, banco de dados) e que são distribuídos como uma *virtual appliance* (baseado no padrão *Open Virtualization Format* – OVF [DMTF 2009]) em nuvens híbridas. *Virtual Appliance* corresponde a uma nova forma de "empacotar" um serviço de software através de conjunto de máquinas virtuais (cada uma contendo sua pilha de software específica) e metadados que descrevem diversas informações como, por exemplo: (i) a capacidade de cada máquina virtual (CPU, memória, disco, rede com a qual sua interface está conectada, etc.); (ii) seqüência de startup das máquinas virtuais; (iii) conjunto de pares propriedades-valor a serem configurados em tempo de *boot* das máquinas virtuais. O Padrão OVF é uma especificação aberta (baseada em XML e XML Schema) para criar *virtual appliances* que está sendo suportada pelas grandes empresas na área de computação em nuvem e virtualização como VMware, Citrix, IBM, Sun, Cisco, Microsoft, dentre outras.
- A distribuição de cada componente da aplicação se faz de forma automática, através de chamadas às APIs baseadas em um padrão genérico de API para nuvens chamado *Open Cloud Computing Interface* - OCCI [OCCI 2009]. A implementação funciona como um adaptador que encapsula chamadas correspondentes às APIs proprietárias de nuvens de infra-estrutura distintas.
- Em tempo de inicialização de cada componente, ou seja, quando sua máquina virtual é inicializada, a aplicação tem o poder de se auto-configurar através de scripts (ex: servidor de aplicação configura o IP do banco de dados ao qual ele referencia).

O restante deste artigo é organizado da seguinte forma: a seção 2 apresenta uma visão geral sobre o estado da arte da computação em nuvem; a seção 3 detalha a abordagem Uni4Cloud e seus componentes arquiteturais, os quais são validados através de um exemplo discutido na seção 4; a seção 5 apresenta os trabalhos; por fim, a conclusão e trabalhos futuros são apresentados na seção 6.

2. Visão geral sobre computação em nuvem

A computação em nuvem é um modelo computacional que permite o acesso, sob demanda e através da rede (geralmente na internet), a um *pool* de recursos computacionais (redes, servidores virtuais ou físicos, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados. Algumas características do modelo de nuvem são [Leavitt 2009, Mell and Grance 2009]:

- Atendimento *self-service* e sob demanda. Um consumidor de um serviço de nuvem pode requisitar automaticamente (normalmente usando-se de interfaces de programação ou APIs) um recurso computacional (por exemplo, um servidor virtual ou espaço em um servidor de armazenamento em rede).
- Elasticidade. O consumidor do serviço pode requisitar dinamicamente mais ou menos recursos para sua aplicação para se adaptar à demanda dos seus usuários. Por exemplo, em períodos de pico o consumidor solicita à nuvem mais recursos

computacionais (como, por exemplo, servidores adicionais), podendo depois liberar tais recursos, quando os mesmos não forem mais necessários.

- Pagamento pelo Uso e Garantias de serviço (Service Level Agreements – SLAs). Os consumidores pagam aos provedores de serviço de nuvem de acordo com o consumo efetuado (modelo de pagamento pelo uso semelhante a utilidades como energia e gás). A nuvem possui mecanismos para permitir o compartilhamento dos recursos existentes de forma eficiente para os diversos clientes e monitorar as aplicações e os recursos de forma a respeitar as garantias de serviço oferecidas como, por exemplo, disponibilidade de 99.9%.
- Acesso ubíquo através da rede. Os recursos computacionais podem ser acessados através de padrões (como APIs REST baseadas em HTTP ou SOAP) por diversos tipos de clientes (*browsers*, PDAs, celulares) e seus detalhes de funcionamento e complexidades ficam "escondidos" pela nuvem.

Existem atualmente diversos tipos (ou modelos de entrega [Mell and Grance 2009]) de nuvens. A EC2 [EC2 2009], da Amazon, é caracterizada como infra-estrutura como serviço (IaaS), oferecendo máquinas virtuais de capacidades diferentes a preços variados. Por exemplo, a menor máquina virtual oferecida pela EC2 (*Small Instance*) possui uma CPU de aproximadamente 1,2GHz, memória RAM de 1,7 GB e 160 GB de armazenamento em disco, e custa 10 centavos de dólares por hora. Outro tipos de nuvens conhecidos são plataforma como serviço (PaaS) e software como serviço (SaaS), que oferecem, respectivamente, plataformas de desenvolvimento e aplicações como serviço. Exemplos comerciais de PaaS são as plataformas Google App Engine [Appengine 2009] e Microsoft Azure [Azure 2009], que oferecem ambientes de desenvolvimento de aplicações em nuvem. Um exemplo bem conhecido de SaaS é Salesforce.com [Salesforce 2009], que oferece diversas aplicações (software comerciais como *CRM*, *ERP*, etc.) como serviço na sua própria infra-estrutura de nuvem.

Conforme mencionado anteriormente, a computação em nuvem adota alguns conceitos e abordagens de áreas já consolidadas como a computação em grade (*grid computing*); porém, é importante ressaltar que existem algumas diferenças tais como [Foster et al. 2009]:

- Modelo de negócio: o modelo de computação em nuvem é um modelo de pagar pelo uso similar a utilidades como eletricidade. Os provedores de serviços de nuvem se beneficiam de economias de escala para realizar lucros e oferecer seus serviços a um baixo custo para os clientes. A computação em grade é normalmente um modelo mais voltado para a pesquisa que envolve a cooperação de grandes centros de pesquisa do governo e universidades. A ideia da grade (*grid*) também consiste de compartilhar recursos para que os pesquisadores submetam processos (*jobs*) para resolver problemas computacionais complexos (ex: projetos na área genética ou de previsão de tempo, simulação, etc.). Normalmente a computação em grade oferece tais recursos computacionais para os membros participantes de um determinado projeto.
- Arquitetura e distribuição: normalmente em grades os recursos computacionais compartilhados (e.g., máquinas) podem estar em regiões geográficas e pertencer a instituições diferentes. Isto em geral não ocorre na computação em nuvem,

cujas soluções são normalmente oferecidas pela mesma empresa embora também se possa haver distribuição geográfica (a Amazon, por exemplo, oferece seus serviços na costa leste e oeste dos EUA e na Europa). Outra diferença é que a distribuição de aplicações em nuvem dá-se normalmente através de máquinas virtuais, enquanto que em grades estas normalmente consistem de processos submetidos para execução em *batch*.

- Tipos de aplicação: a computação em grade é normalmente voltada para o processamento *batch* de problemas de alto desempenho, enquanto que as aplicações em nuvem são normalmente voltadas para aplicações comerciais na Internet. A computação em nuvem, em princípio, também pode ser uma alternativa interessante para aplicações de alto desempenho como no exemplo da aplicação do *NY Times* descrita anteriormente.

Um grande desafio dos atuais clientes de nuvens de infra-estrutura é de possuir um mecanismo que facilite a modelagem, implantação e configuração de uma aplicação que possa ser usada em nuvens distintas. Normalmente, cada provedor IaaS possui um mecanismo proprietário para empacotar e implantar os componentes da aplicação em máquinas virtuais que são implantadas com chamadas à APIs proprietárias.

Este artigo mostra uma abordagem para lidar com a distribuição de aplicações complexas sobre nuvens de infra-estrutura que se baseia em padrões e soluções open source que permitem a distribuição dos componentes da aplicação em diferentes nuvens de infra-estrutura.

3. Abordagem Uni4Cloud

A abordagem Uni4Cloud (ver Figura 1) visa facilitar a modelagem, implantação, configuração e distribuição de aplicações em nuvens de infra-estrutura distintas. A abordagem considera três principais *stakeholders*:

- *Provedor de infra-estrutura de nuvem*. Oferecem serviços de "aluguel" de infra-estrutura como máquinas virtuais (contendo CPU, memória RAM, disco e acesso a rede). Exemplos são Amazon EC2, GoGrid e também a nuvem acadêmica da Universidade da Califórnia em Santa Barbara. Na Figura 1 existem dois exemplos de provedores de IaaS (Amazon e nuvem da Empresa X). A abordagem Uni4cloud se foca em integrar ferramentas e software para facilitar a distribuição de aplicações sobre IaaS distintas.
- *Provedor de serviços web*. Utiliza-se de um provedor de IaaS para distribuir sua aplicação web visando uma redução de custos com infra-estrutura. O objetivo do provedor de serviço é oferecer seu serviço (e.g., um site de vendas on-line como o Provedor de Serviços 1 ou um site de Cursos Online como o Provedor de serviços 2) da forma mais eficiente e econômica possível para seus clientes web. O Provedor de Serviços 1, por exemplo, possui um site de vendas cujos componentes de software são distribuídos em nuvens diferentes (Componente 1 na sua nuvem privada e Componentes 2 e 3 na Amazon). Isso pode ser feito da forma mais simplificada possível através do auxílio da ferramenta de configuração do serviço web que permite a modelagem, configuração e

implantação dos componentes da aplicação através da criação de uma *virtual appliance* usando a abordagem Uni4Cloud.

- *Clientes web (usuários).* Acessam os serviços (através da internet usando *browsers*, PDAs, celulares, etc.) do provedor de serviço (e.g., loja online) sem saber detalhes do seu funcionamento e de onde a aplicação está sendo hospedada. Portanto, para os clientes não importa como o serviço está distribuído (e.g., usando uma ou mais nuvens). Note que o site de vendas online é composto de 3 componentes (máquinas virtuais), sendo que 2 destes componentes contém o servidor web (Apache) e de aplicação (JBoss) e o outro componente possui o banco de dados (MySQL Server)[1]. Cada máquina virtual pode conter um sistema operacional distinto e ser distribuída (de forma transparente para os clientes web) em qualquer um dos servidores físicos que fazem parte de uma ou mais nuvens.

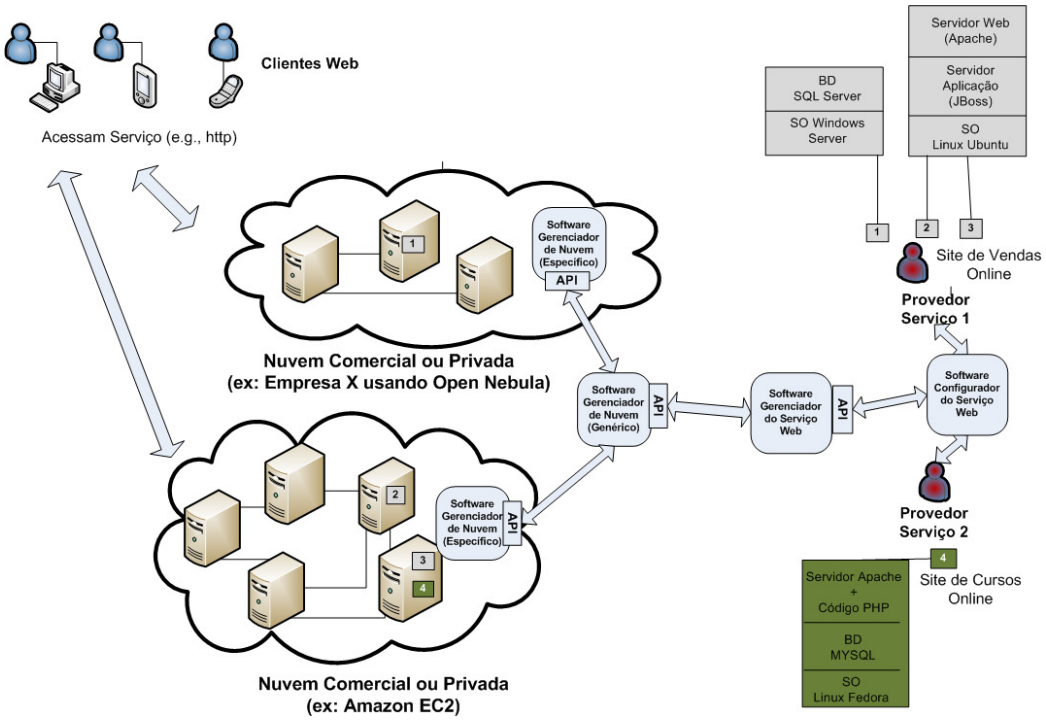


Figura 1 – Abordagem Uni4Cloud

Para dar o suporte mencionado acima, a Abordagem Uni4cloud provê os seguintes componentes[2] da sua arquitetura:

- *Software Configurador do serviço web.* Esta camada é responsável pela especificação do serviço web e das máquinas virtuais que o compõem. Uma ferramenta visual permite a modelagem da *virtual appliance* especificando as máquinas virtuais necessárias e quais componentes de software estão contidos

[1] O site de vendas online neste exemplo distribui a máquina virtual contendo o servidor de banco de dados na sua nuvem interna por razões de segurança. Ele também distribui duas máquinas virtuais contendo o servidor web e de Aplicação dentro da nuvem da Amazon EC2 e pode se beneficiar do conceito de elasticidade para lidar com a demanda dos usuários balanceando a carga para estes servidores replicados.

[2] A tecnologia usada para implementar estas camadas de software segue a filosofia de Web Services (REST ou SOAP) e se utiliza de padrões e soluções open source já existentes.

nelas além das suas propriedades de configuração. Esta ferramenta é uma extensão do software *open source* Jasmine Design [Jasmine 2009] que gera a especificação do serviço web baseada no padrão OVF [DMTF 2009] à partir do modelo visual montado da *Virtual Appliance*.

- *Software gerenciador do serviço web.* Esta camada oferece mecanismos para ler a especificação do serviço fornecida pelo configurador e fazer uma "tradução" (i.e., o parsing) para chamadas específicas do software gerenciador de nuvem de forma a distribuir os componentes de software em uma ou mais nuvens de acordo com a especificação. Além disso, esse software permite monitorar o serviço web, ou seja, a aplicação do cliente que é executada em uma ou mais máquinas virtuais da nuvem. Este serviço também oferecerá mecanismos que poderão auxiliar a aplicação a tomar decisões dinamicamente, como, por exemplo, fazer o *start up* de mais um servidor de aplicação devido ao aumento da carga do sistema (devido a um pico de uso, por exemplo).
- *Software gerenciador de nuvem (genérico).* Este software funciona como um adaptador que se baseia na API genérica de software gerenciador de nuvem OCCI e traduz as chamadas feitas para o gerenciador de nuvem específico correspondente. Na prática, são implementados vários adaptadores (um adaptador para cada nuvem IaaS distinta). A vantagem de usar estes adaptadores é tornar o software gerenciador de serviço web fracamente acoplado a uma nuvem IaaS específica.
- *Software gerenciador de nuvem (específico).* Este software gerencia e controla a alocação das máquinas virtuais nos servidores físicos pertencentes à nuvem IaaS. É importante ressaltar que este software **não é** um componente implementado pela Uni4Cloud, mas sim por um provedor IaaS qualquer (ex: Amazon EC2) que oferece uma API proprietária com diversas funcionalidades de gerenciamento do ciclo de vida das máquinas virtuais (*start*, *shutdown*, *describe*, etc.).

A Figura 2 mostra uma visão geral de um cenário básico de implantação de um serviço web em uma ou mais nuvens com suporte da abordagem Uni4Cloud.

O *stakeholder* provedor do serviço web utiliza-se do **configurador de serviço web** para modelar e configurar a *Virtual Appliance* conforme mostrado na Figura 3. O configurador irá permitir montar a *Virtual Appliance* através de componentes reutilizáveis (e.g., uma aplicação baseada em um cluster de servidores de aplicação J2EE contendo dois servidores JOnAS e um servidor web Apache configurado também para balanceamento de carga e o banco de dados) e gerar a especificação OVF correspondente (um arquivo XML que pode conter referencias a outros arquivos como os arquivos de imagem de cada máquina virtual).

Uma vez que o serviço esteja configurado, o usuário (no caso o provedor de serviços web da Figura 1) pode distribuir seu serviço invocando a API de distribuição de serviço e que recebe a especificação OVF do serviço como parâmetro para o **gerenciador de serviços web.** Este irá fazer o parsing da especificação OVF (do arquivo XML) e traduzir isso em chamadas correspondentes à inicialização das máquinas virtuais de acordo com a sua especificação OVF. Isso será feito para todas as máquinas virtuais contidas na especificação OVF e se traduzirá em chamadas a API do Gerenciador de Nuvem genérico.

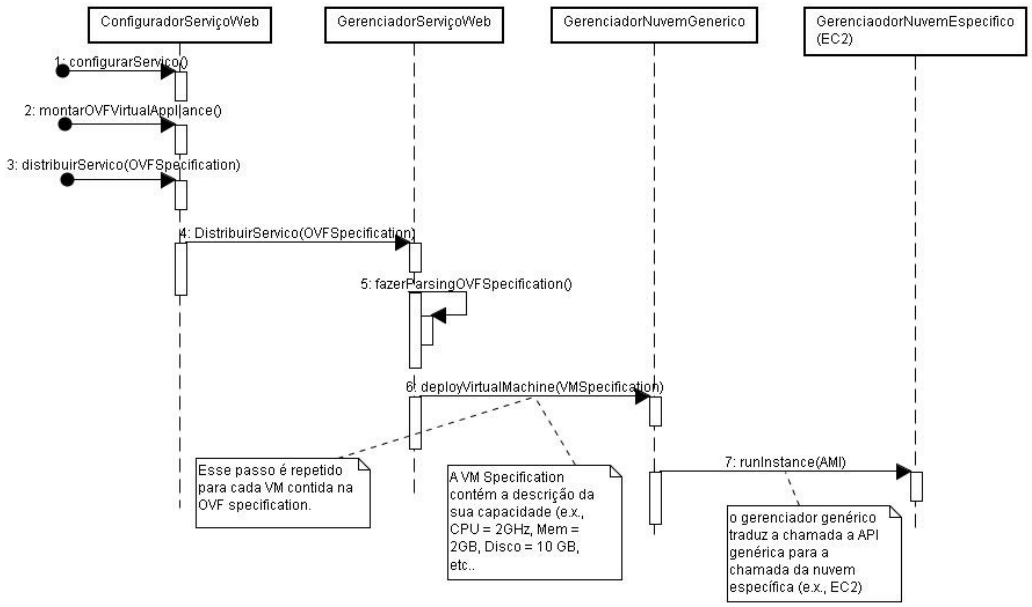


Figura 2 - Visão geral da implantação de um serviço web

O gerenciador de nuvem genérico se baseia na especificação do padrão OCCI [OCCI 2009] e representa um adaptador genérico para traduzir as chamadas para uma determinada nuvem específica em tempo de execução. Isto é importante, pois as máquinas virtuais contidas dentro da especificação OVF podem ser distribuídas em nuvens distintas. Uma vez que o gerenciador de nuvem genérico obtenha a informação de qual(is) nuvem(s) específica(s) será(ão) usada(s), ele irá fazer a chamada correspondente como, por exemplo, chamar o serviço *runInstance()* do gerenciador de nuvem específico da EC2 (ver Figura 2).

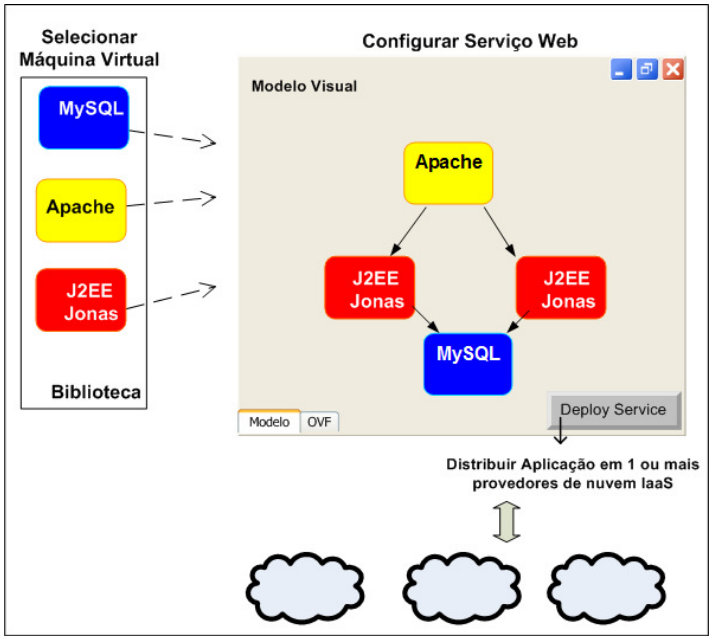
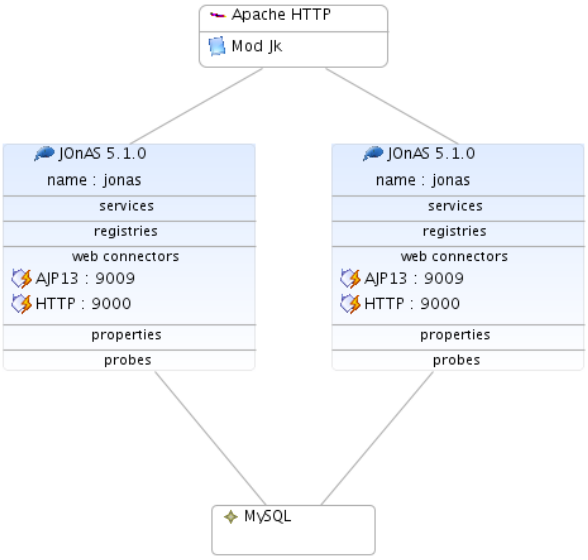


Figura 3 - Ferramenta de configuração e modelagem do serviço web

4. Utilização da Abordagem

A abordagem Uni4Cloud foi utilizada na prática com o objetivo de investigar como ela pode dar suporte à implantação automática de aplicações web em nuvens distintas. A aplicação web escolhida foi uma *Virtual Appliance* para comércio eletrônico, composta por quatro máquinas virtuais: uma com o servidor web *Apache*, que também foi configurado como balanceador de carga, duas executando o servidor de aplicação *Java EE JOnAS 5* em *cluster* e a última como servidor de banco de dados *MySQL*, conforme ilustrado na Figura 4. A razão da escolha desta arquitetura se deve ao fato de ser um perfil de configuração bastante comum para aplicações web em cluster como sites de comércio eletrônico. O mesmo cenário de validação poderia ter sido realizado com outras tecnologias (ex: servidor de aplicação JBoss, banco de dados Oracle, etc.).

Figura 4 – Virtual Appliance do cluster Jonas



Para implantar a *virtual appliance* da Figura 4 em um conjunto de nuvens de infra-estrutura os passos seguintes são realizados.

(1) Modelagem da *virtual appliance* pelo configurador de serviço web

Nesta etapa um modelo é criado para representar as principais propriedades de cada componente da aplicação. Por exemplo, na Figura 4 cada elemento da figura representa uma máquina virtual que conterá propriedades associadas como: imagem de disco (neste caso, uma imagem Xen de cada VM), nuvem na qual a VM será implantada (neste exemplo usamos duas nuvens privadas configuradas no nosso laboratório, baseadas em Eucalyptus [Eucalyptus 2009] e Open Nebula [Nebula 2009]) além de portas na qual o componente executa. O software configurador do serviço web é uma extensão da ferramenta *Jasmine Design,* a qual incorpora algumas propriedades referentes à computação em nuvem como especificar em qual nuvem o componente irá ser implantado.

(2) Conversão do modelo em uma especificação do Padrão OVF

A Figura 5 mostra parte do arquivo *OVF descriptor* que contém a especificação da aplicação e é gerada automaticamente baseada no cluster de Jonas modelado na Figura 4. O arquivo OVF contém as máquinas virtuais citadas acima, quatro imagens contendo o sistema operacional *Debian GNU/Linux Etch*, além das configurações de rede. A Figura 5 mostra a seção <*References*> que faz referência às quatro imagens utilizadas que foram armazenadas em um repositório de imagens no nosso laboratório. A seção <*ProductSection*> mostra o exemplo de uma propriedade que é o endereço IP da VM que contém o Apache e que é gerada dinamicamente em tempo de implantação por DHCP. A seção <*VirtualSystemCollection*> contém as especificações de cada máquina virtual <*VirtualSystem*> como nós filhos. Por exemplo, a *virtual system* que representa a máquina virtual que contém o Apache é parcialmente mostrada na parte inferior da figura especificando o hardware daquela VM (CPU de 2 cores, 512 Mb de RAM, disco e interface de rede).

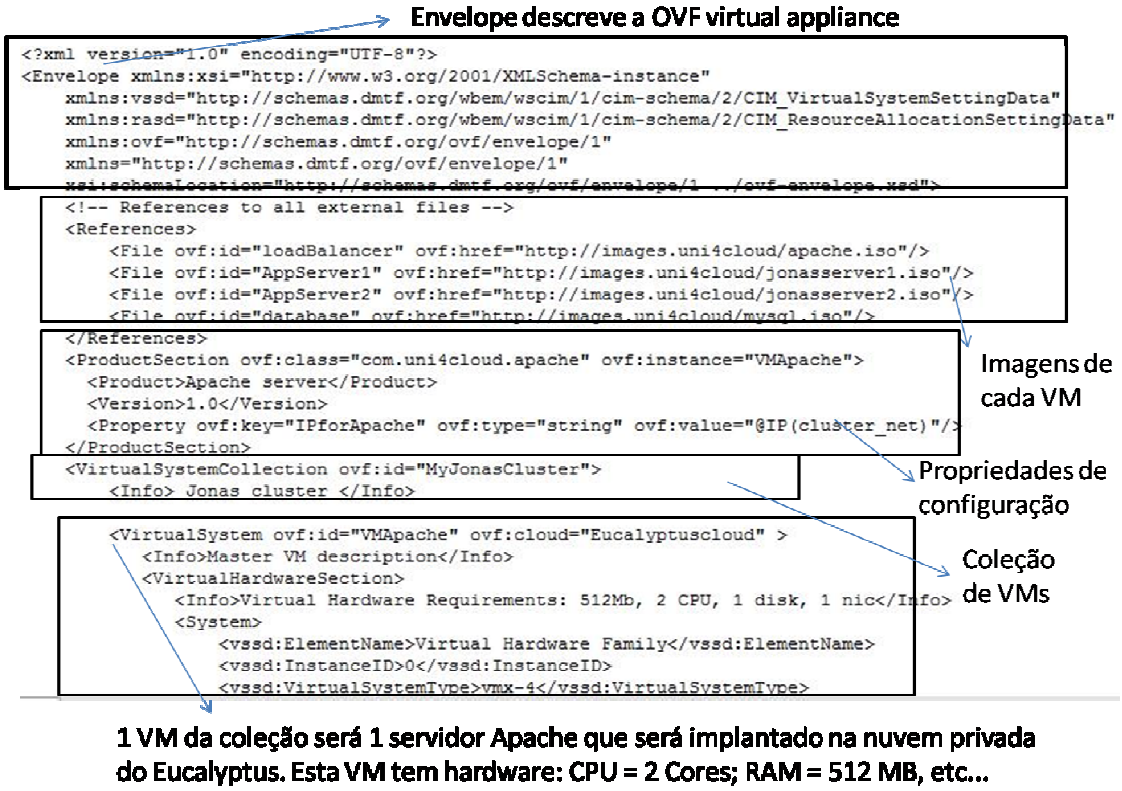


Figura 5 – *Virtual Appliance* do cluster JOnAS no Formato OVF

(3) Distribuição automática da aplicação

Uma vez que o arquivo OVF é gerado pelo **configurador de serviço web** pode-se solicitar a implantação da virtual appliance o que resulta na invocação do **gerenciador de nuvem** que irá interpretar o arquivo *OVF descriptor* da Figura 5 para implantar as VMs nas nuvens IaaS escolhidas. Neste exemplo, o gerenciador genérico de nuvem tinha registrado dois adaptadores para os **gerenciadores de nuvem específicos para *OpenNebula* e para *Eucalyptus*,** que são responsáveis por fazer as chamadas às APIs específicas de cada nuvem IaaS (ex: chamar o serviço *runInstance()* do Eucalyptus para

implantar a VM do Apache na nuvem do Eucalyptus). Ao final do processo, o novo serviço é disponibilizado, distribuindo cada máquina virtual na sua devida nuvem, com todos os componentes configurados (cada VM ao fazer o *boot* pela primeira vez pode realizar configurações locais baseadas em scripts como, por exemplo, configurar o IP do banco de dados ao inicializar as VMs de cada servidor JOnAS).

A Abordagem Uni4Cloud propicia os seguintes benefícios:

- Possibilidade de trocar o provedor IaaS de forma simples bastando alterar o modelo e realizar uma nova implantação. Isto beneficia empresas que tenham interesse em disponibilizar sua *virtual appliance* em diversas nuvens distintas dependendo de qual lhe traga mais benefícios (ex: custo menor, maior segurança, etc.). Isto também ajuda a minimizar o problema de ficar preso a um provedor IaaS específico (*vendor lock in*).
- Utilização de padrões da área de *cloud computing*. O padrão OVF está sendo adotado em soluções de *cloud computing* de grandes empresas como VMWare IBM, Microsoft, Sun dentre outras para construção de *virtual appliances*. Basear-se neste padrão facilita integrar a abordagem Uni4Cloud com soluções destas empresas.
- Facilitar a implantação automática de aplicações web. O exemplo baseado no cluster de JOnAS descrito anteriormente mostra como a Uni4Cloud pode facilitar a implantação de uma aplicação relativamente complexa em nuvens de IaaS. Tal cenário poderia ser bastante útil na prática, pois empresas consomem bastante tempo (e dinheiro) para instalar e configurar aplicações com este perfil em servidores físicos em seus *data centers* locais. A Uni4Cloud provê uma forma simples de distribuir a aplicação automaticamente em nuvens IaaS distintas, abstraindo detalhes da instalação física e simplificando consideravelmente o processo de implantação.

5. Trabalhos Relacionados

A abordagem Uni4Cloud facilita a modelagem e implantação de serviços web sobre nuvens híbridas através do empacotamento da aplicação em uma virtual appliance baseada no padrão OVF. A especificação OVF é interpretada e os componentes da aplicação são implantados automaticamente em nuvens IaaS distintas.

O projeto Europeu Reservoir [Rochwerger et al. 2009] se baseia em um conceito de federação de nuvens onde vários provedores IaaS (que são membros da federação Reservoir) implementam a mesma camada de middleware (ex: algo similar ao software gerenciador de nuvem específico) que dá suporte ao funcionamento da sua nuvem IaaS. Estes provedores são capazes de se comunicar entre si e cooperar (ex: migrar uma máquina virtual de um provedor para outro) pois se baseiam na mesma tecnologia do middleware de nuvem Reservoir. No entanto, o projeto Reservoir ainda não suporta a distribuição de um serviço web de forma integrada com outros provedores IaaS (como GoGrid, Amazon EC2, etc..) e também não facilita a modelagem do serviço para criação automatizada das *Virtual Appliances*. Portanto, a abordagem Uni4Cloud oferece funcionalidades complementares ao Reservoir, podendo escolher o Reservoir como uma

potencial nuvem de IaaS na qual um determinado componente de software pode ser distribuído.

O projeto Altocumulus, da IBM [Maximilien et al. 2009], propõe uma plataforma de middleware que oferece uma API uniforme para implantar e gerenciar recursos e aplicações em nuvens heterogêneas. Os serviços oferecidos pela API do Altocumulus são mapeados para os serviços equivalentes oferecidos por diferentes provedores de nuvem através de adaptadores criados especificamente para cada nuvem. Uma limitação do Altocumulus é que ele define a sua própria API genérica, o que pode vir a desestimular a sua adoção por empresas e usuários que não queiram ficar presos a uma API proprietária (API *lock-in*). Ao contrário, a abordagem Uni4Cloud se baseia no padrão OCCI que está sendo suportado por diversos provedores de nuvem IaaS. Outra limitação é que o Altocumulus atualmente não suporta a modelagem da virtual appliance e sua implantação de forma híbrida, ou seja, contendo diferentes componentes da mesma virtual appliance em nuvens distintas (no Autocumulus todas as máquinas virtuais da mesma *virtual appliance* são implantadas na mesma nuvem IaaS).

6. Conclusão e Trabalhos Futuros

Este artigo apresentou a abordagem Uni4Cloud que procura facilitar a modelagem, implantação, configuração e distribuição de aplicações Web complexas em nuvens que oferecem servidores virtuais como recurso, chamadas de nuvens de infra-estrutura (IaaS). O artigo dá uma visão geral da implementação dos componentes de software que fazem parte da Uni4Cloud além de mostrar um exemplo de como a abordagem foi usada na prática para implantar automaticamente uma aplicação de comércio eletrônico baseada em um Cluster do servidor de aplicação JOnAS.

Como trabalho futuro pretendemos evoluir a implementação de todos os componentes da arquitetura do Uni4Cloud além de implementar outros componentes relacionados ao monitoramento e auto-adaptação das aplicações (ex: possibilitar que o tamanho do cluster de servidores de aplicação possa crescer/diminuir dinamicamente) e gerenciamento de SLAs.

Também procuraremos realizar estudos de caso para investigar outros exemplos de aplicações (diferentes de aplicações Web, conforme ilustrado neste artigo) como aplicações de alto desempenho e aplicações de *Grid* podem se beneficiar de infra-estruturas de nuvem. Acreditamos que distribuir os componentes de tais aplicações em nuvens IaaS requer processo semelhante ao que mostramos no nosso estudo de caso. Outros projetos discutidos na seção 5, como o Reservoir, já desenvolveram estudos de caso [Galán et al. 2009] em que uma nuvem de IaaS é usada para distribuir os componentes do *middleware* de *grid* da *Sun Microsystems* (*Sun Grid Engine*) e dar suporte a aplicações em grade executando na nuvem.

Agradecimentos

Este trabalho é parcialmente financiado pela Fundação Edson Queiroz, Universidade de Fortaleza, através do Projeto OW2.

Referências

Appengine (2009), Google App Engine, Available from: http://code.google.com/intl/pt-BR/appengine/, March.

Armbrust, M., Fox, A. and Griffith, R. (2009), "Above the Clouds: A Berkeley View of Cloud Computing." University of California Berkley Technical Report. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, March.

Azure (2009), Windows Azure Platform., Available from: http://www.microsoft.com/windowsazure/, March.

DMTF (2009), Open Virtualization Format Specification, Version 1.0.0. Available from: http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf, March.

EC2 (2009), Amazon Elastic Compute Cloud (Amazon EC2), Available from: http://aws.amazon.com/ec2/, March.

Eucalyptus (2009), Eucalyptus - Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems, Available from: http://open.eucalyptus.com/, March.

Foster, Ian T., Zhao, I., Raicu, Ioan, et al. (2009) "Cloud Computing and Grid Computing 360-Degree Compared", In: CoRR abs/0901.0131.

Galán, F., Sampaio, A., Rodero-Merino, L. et al., (2009) "Service Specification in Cloud Environments Based on Extensions to Open Standards". In: Fourth International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE), Dublin, Ireland: IEEE.

Gens, Frank (2008), "IDC consulting. IT Cloud Services Forecast - 2008, 2012, In: A Key Driver of New Growth.

Gottfrid , D., (2007) "Self-service, Prorated Super Computing Fun!", In: New York Times; Available from: http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/, September.

Jasmine (2009), Jasmine, Available from: http://jasmine.ow2.org/, April.

Leavitt, Neal, (2009) "Is Cloud Computing Really Ready for Prime Time?" IEEE Computer, vol. 42, no. 1, p. 15-20.

Lynch, Merrill (2008), "The Cloud Wars: $100+ billion at stake".

Maximilien, E. M, Ranabahu, A., Engehausen, R., Anderson, L (2009), "IBM Altocumulus: A Cross-Cloud Middleware and Platform". In: Proc. of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems, Languages and Applications (OOPSLA'09), pp. 805-806, ACM Press.

Mell, P. and Grance, T. (2009), "National Institute of Standards and Technology, Information Technology Laboratory (NIST) Working Definition of Cloud ", http://csrc.nist.gov/groups/SNS/cloud-computing/index.html.

Nebula (2009), The Open Source Toolkit for Cloud Computing, Available from: http://www.opennebula.org/, March.

OCCI (2009), OGF Open Cloud Computing Interface Working Group, Open Cloud Computing Interface, Available from: http://www.occi-wg.org/, March.

Pettey, C. (2007), Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO2 Emissions. , Available from: http://www.gartner.com/it/page.jsp?id=503867, March.

Pring, Ben and Desisto, Robert P. (2008), "Gartner report ID Number G00162998, Predicts 2009", In: Software as a Service Gains Traction.

Rochwerger, B., Caceres, J., Montero, Ruben S. et al.,(2009) "The RESERVOIR Model And Architecture for Open Federated Cloud Computing." In: IBM Systems Journal

Salesforce (2009), Salesforce.com., Available from: www.salesforce.com, March.

Smith, David M. and Austin, Tom (2008), "Gartner Report ID Number: G00163522, Predicts 2009", In: Cloud Computing Beckons.

VMware (2009), VMware hypervisor, Available from: http://www.vmware.com/, March.

Xen (2009), Xen Hypervisor, Available from: http://www.xen.org/, March.

VIII Workshop em Clouds, Grids e Aplicações

♦

Sessão Técnica 2

Infraestrutura em Aplicações

Porting a Hemodynamics Simulator for a Grid Computing Environment

Paulo G. P. Ziemer, Ramon G. Costa, Pablo J. Blanco, Bruno Schulze, Raúl A. Feijóo

{ziemer, ramongc, pjblanco, schulze, feij}@lncc.br

[1]National Laboratory of Scientific Computing - LNCC
Quitandinha, 25651-075 Petris - RJ, Brazil

Abstract. *The modeling of human cardiovascular system may represent a breakthrough in current way how physicians and researchers understand the general working of cardiovascular system and also could help the development of new treatments to cardiovascular pathologies through the detailed information provided by computer simulation. However, computer simulation of highly detailed models usually demands great computing power and may take long computing time, depending on the size and precision of the model. This way, in order to decrease computing time, the model resolver (SolverGP) uses MPI to share load among different processors. This work presents the main steps of the porting process of SolverGP to a grid environment and also covers the basic characteristics of the middleware gLite, the software tools used in the porting process, such as Watchdog, SecureStorage and MPI-Start and the principal strategies used to enable the execution of SolverGP in the grid environment from EELA-2 project.*

1. Introduction

Diseases related with the human cardiovascular system, such as coronary heart disease, are the leading cause of death worldwide [Mackay et al. 2004]. The use of computer modeling can provide exclusive and detailed insight about the blood flow behavior to Medicals and Researchers, giving them new perspectives about treatments of pathologies and also improve the general understanding of the main physical phenomenons of human cardiovascular system. The computational hemodynamics field presents an increasing need for reproducing accurately physiological blood flow regimes encountered in the cardiovascular system, as well as to simulate coupled global/local phenomenons, with the purpose of retrieving as much information as possible from the numerical simulations [Blanco et al. 2009a].

This way, the HeMoLab project [HeMoLab] was created with the purpose of developing computational tools that could make easier the process of creation, edition and visualization of results of models of the Human Cardiovascular System (HCS). Then, it was developed the software named HeMoLab as an extension of Kitware's visualization software Paraview [Moreland 2009]. HeMoLab allows the creation and customizing of general simplified models (1D models), restricted detailed models (3D), and coupled models (1D-3D). It is also possible to simulate cardiovascular related pathologies, e.g., artery aging process, aneurysms and stenosis, and surgical procedures such as insertion

of stent, creation of artery bypass and etc. The models created/customized by HeMoLab are saved in special format files and used after as input to the software that resolves the simulation (SolverGP). Visualization of produced simulation data is a crucial stage in order to allow a better understanding of main physicals phenomenons that occur within determined model. Thus, several schemas of scientific data visualization have been developed, such as blood particle tracing effects, artery flow 3D visualization with aid of CAVE (Cave Automatic Virtual Environment) and etc.

The experience with the computer simulation of 3D models have shown that the demanded processing time is of the order of weeks or even months in case of more complex models. This way, the use of high performance computing techniques such grid computing is important to the success of adoption of the tool by Medical community. Grid computing [Kesselman and Foster 1998] is a paradigm that proposes aggregating geographically distributed, heterogeneous computing, storage and network resources to provide unified, secure and pervasive access to their combined capabilities. We hope that using grid computing applied to the simulation of cardiovascular models may provide speedup for the execution of computer simulations (since a larger number of processors is available in some grid sites - if compared to the regular number of processors commonly used) and also increase the number of simulations made concurrently.

In order to execute an application in a grid environment, the software must be carefully analyzed by looking for requirements that if not well ”carefully observed” may cause the fail of the execution of the application in the remote site. Examples of restrictions include the use of specific libraries (which may be not ”installed” in remote computers), interaction of application with local Operating System (OS) (for instance, some OSs like Linux allows by default the use of a limited amount of dynamic memory).

The HeMoLab was selected among e-science applications from Latin America to participate of the Second EELA-2 Grid School [Grid-School 2009]. This event aims to instruct how applications that usually runs on centralized computing resources, such as homogeneous local clusters, may be adapted to be executed at decentralized heterogeneous distributed computing resources from EELA-2 project [EELA]. During the school, it was presented the basic and advanced characteristics of the gLite middleware [Meglio 2007] like searching of computing resources, jobs types and specifications and etc.

In this context, this paper describes the main steps of the porting process of SolverGP to gLite environment and also covers the main characteristics of gLite and the tools used during porting process.

2. The HeMoLab project

2.1. Modeling the Human Cardiovascular System

The simulation of the HCS is performed through the use of the following types of models: unidimensional (1D) and three-dimensional (3D). The 1D model represents the cardiovascular system in a simplified fashion, where main arteries of human body are modeled as a set of straight lines (segments) and points (terminals). Segments are used to represent the individual arteries, while terminals are used to model the connection points (between arteries that have different properties) or are used to represent the effects of capillary artery sets. Each segment is sub-divided in elements, called nodes, where the values of blood

pressure, velocity and artery area are calculated during the simulation time. The human heart is also modeled through an element that imposes a regular pressure variation to the whole arterial tree. Each artery has mechanical properties, such as elastin (that indicates how elastic the artery is) and also geometric parameters like artery radius in some given point, wall thickness and etc.

3D models are used to provide a greater level of detail of the blood flow behavior inside a specific artery structure. The geometry of a 3D model may be obtained through the analysis and processing of medical images, such as computer tomography, magnetic resonance, intravascular ultrasound and etc. The use of patient medical image presents the advantage of creation of custom model this way allowing a more realistic hemodynamics study. The artery to be modeled is identified on the image with the use of image segmentation algorithms, such as region growing, topological derivative [Larrabide et al. 2008] and etc. As output of the algorithm, a new image is produced, where the artery area is easily identifiable and then allowing the use of a isosurface algorithm that finally generates the triangle mesh. This mesh usually has not good quality (i.e. triangles not corrected aligned with others - needle shape- or the number of triangles in some surface area is not refined enough to correctly model the artery wall behavior). In order to solve the stated problems, mesh quality improvement algorithms are used in mesh refinement, mesh smoothing, removal of needle triangles and etc. Figure 1, shows an example of a surface model containing cerebral arteries, where one of them presents an aneurysm.

Figure 1. Model Geometry of two cerebral arteries containing an aneurysm.

Once the surface mesh has good quality, it is now possible to obtain the volume mesh using Delaunay triangulation process. The spatial components of velocity and displacement of blood flow and also the value of blood pressure are calculated for each created point inside the surface. The recently created internal structure is named of Volume mesh. After the surface and volume meshes have been created, the mechanical parameters of the model are defined. These include, artery wall type, that can be defined as rigid or deformable allowing the simulation of artery wall movement imposed by the pressure variation (set by the heart). A set of surface triangles can have custom properties, for instance, the modification of value of elastin allows the simulation of aging process in human arteries. At this time, the model is ready to enter the computer simulation stage,

where the results of blood dynamics during simulation time are calculated.

3D models can be used in a standalone way or integrated with 1D models. 1D-3D integrated models are called of Coupled models [Blanco et al. 2009b], where the 3D elements from inflow covers are connected with Terminals or Nodes from 1D model. This way, the simulation result data from the 1D model points, where the 3D model is currently connected with, are used as boundary condition to the 3D model. If a 3D model is used as standalone, then the boundary conditions must be setup in advanced. These must be a constant or variant value of blood pressure or a blood velocity value.

2.2. Simulations Results

After the computer simulation is done, computer graphics techniques can be applied to the produced data and then allow and make easier the process of getting insight about data. In Figure 2, it is possible to see an example of such process on the simulation results of an artery that contains an aneurysm. Streamlines indicate the instantaneous trajectory of blood particles and the value of velocity magnitude. Warping is also used to indicate the value of velocity on the transverse sector of the aneurysm area.

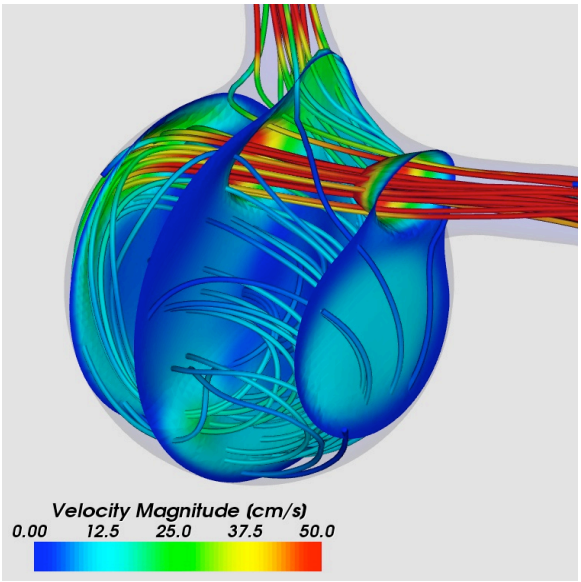


Figure 2. Example of an aneurysm model with simulation results

2.3. General Purpose Solver: SolverGP

The computer simulation of a model is performed through the resolution (using Finite Element Method - FEM) of a Navier Stokes equations system (that describe the motion of fluids, such as blood inside of an artery). Numerical methods are used to provide an approximated result of the analytic result of the equation system. So, these numerical methods were implemented using the programming language Fortran 90 in a system that was named of SolverGP (General Purpose Solver) and this uses the PETSc library [Balay et al. 2008] to make the distributed resolution through MPI (Message Passing Interface) [Forum 1994] of the equation system and also the Parmetis library [Karypis et al. 2003] which is responsible for mesh partitioning.

In order to start a computer simulation, SolverGP reads the set of files that describe the model parameters such as geometrical, mechanical and simulation parameters. Each simulation period (usually one period refers to one heart beat) is sub-divided in time steps (a typical value for it is 640) and for each time step, SolverGP updates the file that stores the simulation results at disk (Dataout.txt) with the values of velocity, displacement and pressure for each volume point. The computing of the simulation can be initialized with some previous calculated result or with a defined state for the model, using for that the file Inifile.txt. This file is updated for each time step and this procedure allows the resuming of a previous computing that has failed.

The SolverGP uses MPICH2 [Gropp et al. 2009] as the MPI standard implementation. MPICH2 programs require that mpd daemons be running in the machines that form the computing processing ring. In Code 1, are shown the needed commands to perform the execution of SolverGP in such environment. In the listed example, the mpdboot command initializes six mpd process in the list of machines specified by the file machine.txt. After that, the mpirun command triggers the process SolverGP and also specifies all the PETSc parameters, such as the maximum number of iterations for the current numerical method (`ksp_max_it`), parameters related with the linear solver in use (`ksp_gmres_restart`, `ksp_rtol`, `ksp_atol`), information about convergence (`ksp_monitor`) and finally the name of log file used to store status information (`run.parallel.log`).

Code 1. Commands used to initialize a SolverGP computer simulation

```
mpdboot -n 6 -f ~./machines.txt
mpirun -np 6 SolverGP.x -ksp_gmres_restart 100 -ksp_max_it 1000 \
-ksp_rtol 1.0e-5 -ksp_atol 1.0e-10 -ksp_monitor -log_summary
</dev/null> & run.parallel.log &
```

The computer simulation of 1D models does not demand high performance computing power. Those simplified models are commonly solved using a sequential SolverGP version on ordinary desktop computers and the solving time is about 10 to 30 minutes. However, more complex models, such as the 3D or coupled models, may demand a high computing power in order to be solved. For instance, the computer model of an aorta, the largest artery of the human body, has approximately 500,000 points and then originating a system with 3,500,000 equations. The computing of just one heart beat period in such model takes approximately 35 days (with 640 time steps) - using eight processors in a 2X quad-core Xeon 3GHz / 64GB RAM machine.

3. Technologies and concepts

During the HeMoLab porting process, several tools were used to support the creation and configuration of grid services, such as Secure-Store Client [Scardaci and Scuderi 2007] and Watchdog [Bruno et al. 2009].

Secure-Store [Scardaci and Scuderi 2007] is used to provide encryption of the files stored at the Storage Elements (SE - grid sites used only for storage). This tool provides command line interface and API for the programming langauges: Java, C++, and etc. Commands are integrated in the gLite User Interface to encrypt/upload and decrypt/download files using the user private key as cryptographic key. The encryption of files is an

important action, since models may contain sensitive information that must be protected from unauthorized access.

The computing of SolverGP may fail (diverge) if simulation parameters are not correctly set. Once that happen the simulation process must be restarted with a new set of simulation parameters. This way, it is necessary that the user keep a certain level of control over how the simulation may develop. In a local cluster, this process is straightforward. However, to monitor files that are located in remote sites is a difficult task since files are not directly available to user. This way, the Watchdog [Bruno et al. 2009] tool was used to provide real time information about remote computing. This tool is a shell script which is included in the JDL (Job Description Language) main script and presents the following features: i) it starts in background before to run the long term job; ii) the watchdog runs as long as the main job; iii) the main script can control, stop and wait until the watchdog has finished; iv) easily and highly configurable and customizable; v) the watchdog does not compromise the CPU power of the Worker Nodes;

Before running a MPI job in a grid Computing environment, one must know some characteristics of remote site, such as, if the remote home disk area is shared between the assigned cluster nodes, the list of machines available (selected by local scheduler) to current session and etc. In order to hide the complexity of establishing the necessary infrastructure to the execution of a MPI job, the set of scripts MPI-Start [HLRS] were developed for the Interactive European Grid Project [I2G]. These scripts supports the initialization of MPI jobs from the following implementations: MPI, OpenMPI, LAM-MPI and MPICH2.

The use of scripts in MPI session configuration allows the user to focus principally in the application related tasks. The number of processing cores is defined through the JDL file and the name of the executable and any parameters are specified using environment variables in the script file MPI-start-wrapper.sh. The functions, pre_run_hook and pos_run_hook from the script file MPI-hooks.sh make possible to specify the previous and post job operations, such as, pre-compiling the MPI code remotely or making some filtering on the data produced after the application has finished its execution.

Beside the present tools, there are several tools to support the creation of a grid environment [JRA1]: Dirac, Transactional Grid Storage Access Framework (GSAF), Grid2Win User Interface, and etc.

4. Aspects related with Software Porting Process

4.1. The structure

The EELA-2 project (E-science Grid facility for Europe and Latin America) aims at building and operating a production-quality grid infrastructure, which provides computing and storage resources from selected partners across Europe and Latin America. Currently, the EELA-2 Consortium encompasses seventy eight member institutions from sixteen countries (five from Europe and eleven from Latin America). GLite [Burke et al. 2009] provides a framework for building grid applications tapping into the power of distributed computing and storage resources across the Internet.

At the time of this writing, twenty four resources centers were connected to the EELA-2 infrastructure and with more than 9600 job slots (CPUs) and around 10 PB of

disk storage [gStat]. As SolverGP requires the use of MPICH2 library, then the number of Computer Elements (CE - grid computer node that actually runs the job) that can be used for the execution of jobs is reduced. The Table 1 shows CEs that are suitable for HeMoLab simulation jobs.

Table 1. list of sites that can run jobs from HeMoLab

CE	Number of Cores
ce-eela.ciemat	216
ce01.eela.if.ufrj.br	216
ce01.unlp.edu.ar	10
eelaCE1.lsd.ufcg.edu.br	8
grid001.cecalc.ula.ve	48
grid012.ct.infn.it	200
gridgate.cs.tcd.ie	715

4.2. The Architecture

A service layer was created between the SolverGP and the services provided by EELA-2. The Figure 3 illustrates the proposed architecture:

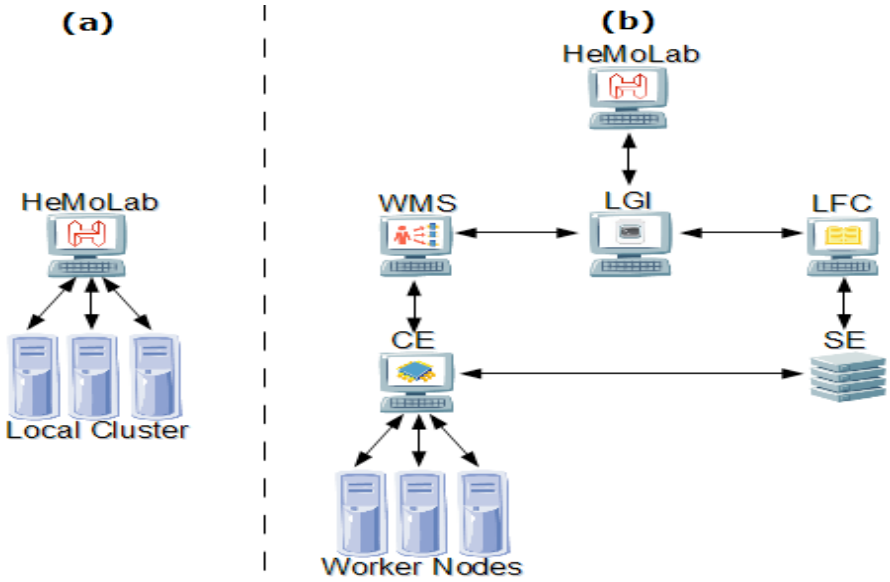


Figure 3. (a) The standard HeMoLab architecture; (b) The proposed architecture that integrates HeMoLab and the services of the EELA-2

The Local Grid Integration (LGI) is the component that retrieves input files and also selects the grid site that best matches with SolverGP execution requirements. At the selected CE, scripts are executed (prior the execution of simulator) in order to correctly configure the environment of the Worker Nodes. The LGI interacts with the Workload Management System (WMS) which has set of grid middleware components that are responsible for the distribution and management of tasks across grid resources.

The SE and LCG File catalog (LFC) are components that provides the storage services: storage of big files (files sent and received through JDL have a maximum size

limitation) and availability (the use of replicas is allowed)/ localization of data files (catalog contains the logical to physical file mappings).

Summarizing, the LGI is responsible for:

1. creation of the interface between HeMoLab and CE - used to launch the remote script that executes the SolverGP;
2. encryption of input files and decryption of output data through the use of private key found at User's digital certificate;
3. triggering of jobs monitoring, which is required by HeMoLab in order to monitor the simulation process.
4. receiving the output files generated remotely.

4.3. The Workflow

The Figure 4 shows the workflow in the execution of a HeMoLab numerical simulation as well its interaction with the grid environment services.

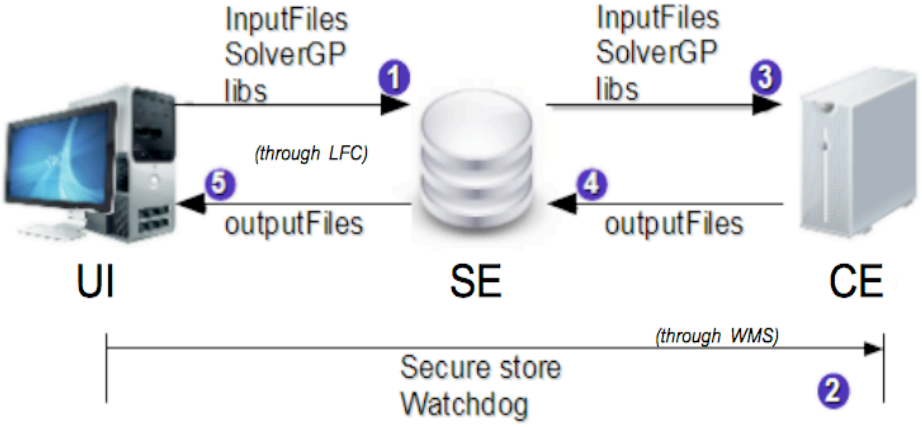


Figure 4. The workflow for a job submission

The User Interface (UI) is the submission machine (grid entry point) and is considered part of the WMS. The LGI is located at UI and contains the JDL file that describes the file names and also details the type of job to be executed. According to the enumeration in Figure 4, follows a brief workflow description:

1. Before submitting a job to remote processing, three files must be stored in SE: i) inputFiles - files generated by HeMoLab that contain model and simulation information required for the numerical solver; ii) SolverGP - the executable file of the SolverGP (responsible for the numerical simulation); iii) libs - extra libraries required by the execution of SolverGP (only used if SolverGP is linked using shared libraries). The input files sent to the SE are encrypted locally in the UI, using the Secure Storage-client and the encryption key is the user's private key.

2. A JDL file defines the files to be sent to CE through WMS. The parameters inputSandBox and outputSandBox lists the files to be sent and received by UI, respectively. Three files are sent to CE through the WMS: i) solver.sh bash script responsible for the downloading of files stored in the SE and also launchs the execution of the SolverGP; ii) watchdog - files used to enable the online monitoring of the execution steps in the numerical simulation; iii) secure store - libraries required to decrypt the input files.

3. the remote script is executed, the files are moved from SE to CE, the numerical simulation is launched and finally the output data is generated. At the moment that the output data is being generated, it is possible to monitor the convergence of the simulation process, through the use of a Watchdog session and this way monitor problems that may be occurred in the numerical simulation, or check the output files for immediate analysis.

4. The output files are sent to the SE and the log files are returned to the UI through WMS.

5. Output files are now moved from SE to the local UI.

5. Conclusions and Future Work

Diseases related to the human cardiovascular system, such as coronary heart disease, are the leading cause of death worldwide. This way, the modeling of the behavior of such system may provide us a better understanding about how patologies develop and this information may be used, for instance, in the development of new treating procedures. The modeling process can make use of simplified general models (1D), restricted detailed models (3D), and coupled models (1D-3D). Those kind of models allow to perform study of cardiovascular related pathologies, e.g., artery aging process, aneurysms and stenosis, and the simulation of surgical procedures such as insertion of stent, creation of artery bypass and etc. However, the numerical simulation of 3D or Coupled models have shown that the demanded processing time is of the order of weeks or even months in a more complex models. This time factor definitely is a limiting issue to spreading the use of HeMoLab tool. This way, the use of grid computing is very important since that the expected improvements of simulation process may help the adoption of the tool by Medical community.

This paper presented the main steps performed to allow the execution of SolverGP in EELA-2 computer grid and also the tools used in this process, such as watchdog - that is used to extract ”on the fly” information from remote simulation- and SecureStorage - which allows the encryption of the sent data to remote grid sites. During grid school, only small models (that are solved quickly) were tested in order to allow the quick reply of the submitted jobs. As future work, it is expected the execution of more computing intense simulations needing for that the increase of the number of processors. Another very important work to do is the specification of the optimum number of processors used for the resolution of a given model, i.e., to make a deeper study of the speedup curve of SolverGP. The expected output from this study is to know in advanced the number of processors that faster solves a model with a specific size. It is also expected to use the gLite programming API [gLite 3.1 API Usage Examples] in order to allow the submission and

monitoring of jobs directly from the HeMoLab application (client side). This brings the use of grid technology to users that are not aware that the simulation is actually being performed in a remote computing grid.

Among the main contributions, we can highlight the possibility of simulating multiples jobs in a parallel way, one in each remote cluster. The number of clusters that supports MPICH2 applications is expected to increase in the near future. It is being analyzed the possibility of dividing a single simulation among several clusters, allowing this way the explicit parallelization of simulation sectors that are data independent.

As main result of this initial work, is expected that the use of the grid enabled application may spread the use of HeMoLab tool among Brazilian medical research community, since a larger number of simulations are expected to be executed at same time and also to obtain smaller times from the execution of the simulations. In order to verify the possible improvements, further work and tests shall be performed.

References

Balay, S., Buschelman, K., and et. al. (2008). *PETSc - Portable, Extensible Toolkit for Scientific Computation Users Manual*. Argonne National Laboratory, US. `http://www.mcs.anl.gov/petsc/petsc-as/`.

Blanco, P. J., Pivello, M. R., Urquiza, S. A., and Feijóo, R. A. (2009a). Building coupled 3d–1d–0d models in computational hemodynamics. In *1st International Conference on Mathematical and Computational Biomedical Engineering - CMBE2009*, Swansea, UK.

Blanco, P. J., Pivello, M. R., Urquiza, S. A., and Feijóo, R. A. (2009b). On the potentialities of 3d-1d coupled models in hemodynamics simulations. *Journal of Biomechanics*, 42(7):919–930.

Bruno, R., Barbera, R., and Ingra, E. (2009). Watchdog: A job monitoring solution inside the eela-2 infrastructure. In *Proceedings of the Second EELA-2 Conference*, Choroni, VE.

Burke, S., Campana, S., and et. al. (2009). *gLite User Guide*. `http://glite.web.cern.ch/glite/`.

EELA. E-science grid facility for europe and latin america. `http://www.eu-eela.eu/`.

Forum, M. P. (1994). Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA.

gLite 3.1 API Usage Examples. Mpi-start. `https://grid.ct.infn.it/twiki/bin/view/GILDA/APIUsage`.

Grid-School (2009). Second eela-2 grid school. `http://indico.eu-eela.eu/conferenceDisplay.py?confId=200`.

Gropp, W., Lusk, E., and et. al. (2009). *MPICH2 - Message-Passing Interface (MPI) Implementation Users Guide*. Argonne National Laboratory, US. http://www.mcs.anl.gov/research/projects/mpich2/.

gStat. gstat monitoring page - grid information system. from:http://goc.grid.sinica.edu.tw/gstat/eela/.

HeMoLab. Hemodynamics modeling laboratory. http://hemolab.lncc.br.

HLRS. Mpi-start. http://www.hlrs.de/organization/av/amt/research/mpi-start.

I2G. Interactive european grid project. http://www.i2g.eu/.

JRA1. Eela jra1: Development of services for applications and infrastructure. http://glite.web.cern.ch/glite/documentation/.

Karypis, G., Schloegel, K., and Kumar, V. (2003). *ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering*. University of Minnesota, Department of Computer Science and Engineering. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.

Kesselman, C. and Foster, I. (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

Larrabide, I., Feijóo, R. A., Novotny, A. A., and Taroco, E. A. (2008). Topological derivative: A tool for image processing. *Comput. Struct.*, 86(13-14):1386–1403.

Mackay, J., Mensah, G. A., Organization, W. H., for Disease Control, C., and Prevention (2004). The atlas of heart disease and stroke - deaths from coronary heart disease. *World Health Organization, Geneva*, page 112.

Meglio, A. D. (2007). The egee glite middleware. *Second Meeting of the Spanish Thematic Network on Middleware for Grids, Barcelona, Spain (2007)*.

Moreland, K. (2009). *The ParaView Tutorial - A Parallel Visualization Application*. Sandia National Laboratories. http://www.paraview.org/.

Scardaci, D. and Scuderi, G. (2007). A secure storage service for the glite middleware. *International Symposium on Information Assurance and Security*.

Uma Infraestrutura para Experimentos Robóticos Bio-Inspirados em Grades Colaborativas

Lucio Agostinho[1], **Ricardo S. Souza**[1], **Fernando Paolieri**[1], **Eliane G. Guimarães**[2], **Eleri Cardozo**[1]

[1]Faculdade de Engenharia Elétrica e de Computação – FEEC
Universidade Estadual de Campinas – UNICAMP

[2]Centro de Tecnologia da Informação Renato Archer
13083-970 – Campinas - SP

Abstract. *This paper presents an infrastructure for robotic bio-inspired experiments in Computational Grids. This infrastructure supports collaboration for groups of resources that can interact among them in a decentralized model, according to needs of the aggregated resources. An experiment in mobile robotics employing an evolutionary navigation algorithm and a second experiment involving a wireless sensor network are presented in order to illustrate the benefits of the proposed infrastructure.*

Resumo. *Este artigo apresenta uma infraestrutura para experimentos robóticos bio-inspirados em Grades Computacionais. Essa proposta suporta colaboração para grupos de recursos que podem interagir entre si em um modelo descentralizado, de acordo com as necessidades dos recursos agregados. Um experimento em robótica móvel que emprega um algoritmo evolutivo de navegação e um segundo experimento envolvendo uma rede de sensores sem fio são apresentados com o intuito de ilustrar os benefícios da infraestrutura proposta.*

1. Introdução

Uma infraestrutura de Grade Computacional (Grade) oferece um ambiente para compartilhar recursos de armazenagem, processamento de dados e distribuição de tarefas que podem ser processadas por nós geograficamente dispersos. Esses nós compartilham os seus recursos computacionais de forma que tarefas complexas possam ser distribuídas e processadas entre os recursos agregados. A Grade pode ser utilizada tanto para a computação de tarefas complexas que exigem alto poder de processamento quanto para a integração de domínios [Foster and Tuecke 2005] [OGF 2010].

A gerência desses recursos pode ser simplificada com o uso de *toolkits*, que são bibliotecas de software que oferecem serviços para manter o ambiente funcional. Os *toolkits* simplificam o acesso à rede de recursos e possuem interfaces para a comunicação com o ambiente. O *grid middleware* é o software que une a aplicação aos recursos e permite o gerenciamento do sistema [IT Professional 2004]. Muitas aplicações podem compartilhar os recursos oferecidos que, em muitos casos, encontram-se dispersos. A provisão deles depende principalmente de mecanismos de autorização, autenticação, agendamento, visualização de status, contabilidade de uso, entre outros. Os usuários desse ambiente precisam dispor de interfaces compatíveis para interagirem

com o ambiente da Grade. Por questões de segurança a agregação de recursos de domínios diferentes irá depender do acordo estabelecido entre os nós da Grade, o que geralmente é feito com o uso de certificados digitais assinados por Entidades Certificadoras (*Certificate Authority*-CA).

No tráfego de informações inter-domínios deve existir a liberdade de escolha da infraestrutura de *software* mais adequada para cada ambiente. Em virtude disso, a evolução da tecnologia de Grades tornou possível a oferta de serviços sob demanda [Cirne and Neto 2005]. A convergência de tecnologias da área de computação de alto desempenho e de padrões abertos trouxe a necessidade de definir um padrão comum e aberto para aplicações baseadas em Grade [Sotomayor 2005]. Iniciativas do *The Global Grid Forum* [GGF 2010] levaram ao desenvolvimento de Grades segundo o padrão OGSA (*Open Grid Services Architecture*). OGSA especifica um conjunto de interfaces comuns para grande parte dos serviços oferecidos, o *Web Services Resouce Framework* (WSRF) [OASIS 2010], que é uma infraestrutura para a arquitetura OGSA e que estende a capacidade dos *Web Services*, especificando serviços *stateful*. Ainda que o conjunto de ferramentas de *software* seja baseado em padrões abertos, o desenvolvimento de aplicações distribuídas pode ser oneroso. O *Globus Toolkit* [Globus 2010], por exemplo, oferece um conjunto de interfaces para aplicações em Grade, mas o desenvolvimento de serviços complexos e a representação dos recursos agregados ainda exigem a manutenção de um intrincado conjunto de arquivos [Luo et al. 2009].

Por outro lado, a infraestrutura de Grade também suporta diversos tipos de aplicações colaborativas. Como exemplo, a navegação autônoma bio-inspirada de robôs móveis pode se beneficiar do uso da Grade. Esse tipo de experimento exige a avaliação em recursos físicos para adquirir resultados mais precisos. Apesar das facilidades de se utilizar a simulação, experimentos reais fornecem atrativos de grande relevância, uma vez que as circunstâncias de operação no mundo real são difíceis de serem modeladas em cenários virtuais. Considerando o problema específico da navegação autônoma, resultados mais precisos podem ser obtidos em experimentos evolutivos com robôs reais que utilizam Sistemas Classificadores [Cazangi 2008].

Para essa classe de experimentos um agente pode ser definido como um sistema computacional, situado em um dado ambiente, que tem a percepção desse ambiente através de sensores, tem a capacidade de decisão, age de forma autônoma nesse ambiente através de atuadores, e possui capacidades de comunicação de alto-nível com outros agentes e/ou humanos, de forma a desempenhar uma dada função para a qual foi projetado [Reis 2003]. Embora não exista uma definição consensual desse conceito, essa será a definição adotada nesse artigo.

Afirmar que um agente é autônomo significa que ele consegue se auto-regular gerando as próprias regras que regem a sua atuação no ambiente, além de agir por si só [Siegwart and Nourbacksh 2004]. A navegação diz-se bio-inpirada porque é baseada nos mecanismos evolutivos biológicos [Eiben and Smith 2007]. Nesse processo, a evolução é vista como um mecanismo de busca por soluções para resolver um conjunto definido de problemas. Para isso, geralmente utiliza-se uma função de avaliação que define uma superfície de adaptação. O algoritmo bio-inpirado, proposto neste artigo, realiza um processo iterativo de busca, ou seja, de otimização das soluções encontradas em cada passo da evolução das regras do agente. O algoritmo do Sistema Classificador foi proposto para evoluir as opções de movimentação nesse espaço de busca, conduzindo o agente ao longo do ambiente.

Neste artigo é apresentada uma proposta de infraestrutura para robótica colaborativa em Grade com suporte à agregação dinâmica de recursos ao ambiente. O objetivo é oferecer uma infraestrutura interoperável com o ganho de facilitar a oferta de serviços sob demanda. A arquitetura prevê a agregação segura de serviços associados a recursos. Avalia-se a proposta com a experimentação remota em tempo-real da navegação autônoma bio-inspirada, com o uso da versão 5.0.1 do *Globus Toolkit* (GT). Estudos de caso comprovam a viabilidade da arquitetura nos quesitos de segurança, liberdade de uso da infraestrutura de software no domínio, suporte à colaboração em Grade, controle da distribuição de recursos e desempenho da comunicação no processo de telemetria remota em tempo-real.

O artigo esta organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados com a proposta; a Seção 3 descreve a infraestrutura da Grade, detalhando o modelo e a arquitetura; a Seção 4 apresenta o experimento robótico colaborativo de Sistemas Classificadores na Grade; a Seção 5 avalia o algoritmo evolutivo e o desempenho da arquitetura com um experimento que envolve uma rede de sensores. Finalmente, a Seção 6 apresenta as considerações finais.

2. Trabalhos Correlatos

O estudo de Robótica em Grades ainda é pouco explorado na literatura. Em certo ponto, essa área pode ser vista como um tipo de gerência de recursos de vários tipos de agentes robóticos autônomos inteligentes. A Grade pode fornecer recursos quando os robôs não possuem capacidade de processamento suficiente. Outras capacidades são a tolerância a falhas permitindo a computação redundante, além do compartilhamento de recursos robóticos.

A integração segura de recursos entre domínios geograficamente dispersos é um requisito importante que orientou o desenvolvimento da infraestrutura apresentada nesse artigo. Dentre os trabalhos que abordam essa temática está um projeto de arquitetura Robótica de Grade na Internet [Sabatier et al. 2004], que é descrito como uma alternativa para a integração de dois laboratórios de informática entre a França e a Itália. O projeto utiliza o conjunto de bibliotecas GridRPC [Seymour et al. 2002] para o controle remoto distribuído e redundante de um robô autônomo. Outro projeto de arquitetura Robótica em Grade [Breitling et al. 2008] integra uma rede de telescópios robóticos. Essa rede de recursos é capaz de realizar novos tipos de observações que não podem ser realizadas com instrumentos individuais. É empregado o *grid middleware* AstroGrid-D, que é mantido em uma camada de aplicação acima do *Globus Toolkit*, utilizando os comandos do *Grid Resource and Allocation Manager* (GRAM).

Esse artigo foca na integração de recursos robóticos para experimentos colaborativos. A colaboração diz respeito ao trabalho conjunto em uma atividade, especialmente para produzir ou conduzir uma atividade com dois ou mais parceiros [Parker 2010]. *Grades Colaborativas* consistem de muitos participantes que concordam em compartilhar os seus recursos em um ambiente comum. Nessa Grade os recursos são agrupados em federações sob um mesmo domínio administrativo, e essas federações são comumente referenciadas como Organizações Virtuais (*Virtual Organizations* - VOs). Uma VO é primariamente um domínio gerenciável que controla e coordena os serviços e recursos fornecidos por outros domínios, a fim de atingir um objetivo comum [Guillen 2009]. Os membros de uma VO podem ter acesso a uma variedade de recursos agregados ao ambiente por um determinado período [Foster et al. 2002]. Por outro lado,

a Computação *Peer-to-Peer* (P2P), que é um tipo de Computação Distribuída, pode ser confundida com a Computação em Grade. A Computação P2P diferencia-se da Computação em Grade em razão de seus objetivos serem baseados na oferta de serviços especializados, e estritamente relacionados, para milhares de usuários concorrentes [Foster and Iamnitchi 2003].

Diversos projetos propõem o uso de Grades como suporte para a colaboração. Uma abordagem adotada pelo grupo da *Globus Alliance* é o *Java Commodity Grid (CoG) Kit Ad hoc Grid framework* [Laszewski et al. 2010]. Ele é um protótipo que tem o objetivo de simplificar o uso do *Globus Toolkit* e de utilizar essa ferramenta como suporte para a colaboração entre múltiplos domínios. O projeto faz parte do *Java CoG Kit* e pretende ser um mecanismo para acomodar a colaboração na Grade através de ferramentas de gerência e comunicação entre grupos, provisão de serviços, provisão de QoS, descoberta de serviços, invocação de serviços e políticas de autorização de acesso.

Na criação da infraestrutura apresentada nesse artigo, preocupou-se em minimizar o esforço para agregar recursos robóticos, que devem ser mantidos por um *middleware* com funcionalidades reduzidas no domínio do usuário. Um trabalho mais abrangente, mas que ilustra essa preocupação com o suporte de funções no domínio do usuário, é o Grid4All [Krishnaswamy et al. 2008], que utiliza o termo “grid democrático” para definir a sua proposta. O objetivo é envolver instituições de ensino, pequenas empresas, pesquisadores e outros participantes com PCs menos robustos em uma infraestrutura de computação global que permita a colaboração e o trabalho em conjunto. O cenário foca no suporte ao compartilhamento de informações e na colaboração entre os seus membros. O *middleware* Grid4All possibilita o acesso concorrente às informações, a criação de VOs e a adição de recursos à rede. Por meio de interfaces públicas, a alocação de recursos e o espaço de armazenamento pode ser obtido na própria rede. A infraestrutura é baseada em redes *overlay* com técnicas P2P, auto-organização das redes *overlay* com *Distributed Hash Tables* (DHT) e tolerância a altas taxas de entrada e saáda de recursos (*churn*). A proposta também pretende suportar serviços de dados federados.

Outro requisito definido para a infraestrutura apresentada neste artigo é o de utilizar o ambiente para executar um conjunto de serviços distribuídos, mas mantidos dentro do domínio interno da Grade. O usuário deve ser capaz de criar as suas aplicações, combinando os serviços já oferecidos pelo ambiente. Esse requisito é similar ao do projeto PlanetLab [Peterson et al. 2006] quanto à oferta de serviços distribuídos. O PlanetLab é uma plataforma global para desenvolver e avaliar serviços de rede. O projeto fornece uma rede *overlay* que permite a avaliação e a implementação de serviços distribuídos. A arquitetura atende a pesquisadores que desenvolvem serviços de rede e a clientes que desejam utilizar tais serviços. A execução contínua de aplicações é possível porque uma fatia (*slice*) da rede é atribuída para a execução da aplicação. O mecanismo de virtualização distribuído requer a multiplexação dos serviços concorrentes. A gerência da rede é dividida em serviços de gerenciamento independentes, cada um executando em sua própria fatia da rede [Ziviani and Duarte 2005].

A abordagem de Grades Colaborativas é apresentada em outros projetos como o OurGrid [Andrade et al. 2003] e LaCOLLA [Vilajosana et al. 2007]. A Computação em Grade é utilizada em muitos outros projetos de Computação Distribuída relacionados à Biologia e Medicina, Ciências da Terra, Matemática, Física e Astronomia, Arte, Inteligência Artificial e outros [GRID Infoware 2010].

3. Infraestrutura para Colaboração em Grade

Nossa proposta da Grade Colaborativa foi baseada em um modelo capaz de oferecer serviços sob demanda, otimizar o desenvolvimento de novas aplicações e agregar recursos de diferentes domínios à infraestrutura de serviços já oferecida pelo GT. Como mostra a Fig. 1, um conjunto de módulos foi definido para simplificar a agregação de recursos, mantendo a segurança e a escalabilidade. Esse modelo baseia-se na especificação do CoG Kit [Laszewski et al. 2010] ao definir um conjunto de serviços que executam em uma camada de aplicação superior ao GT. O modelo orientou a criação do *Simple Interface with Grid Agent (SIG Agent)* que é uma interface no domínio do usuário que abstrai o desenvolvimento de aplicações de Grade.

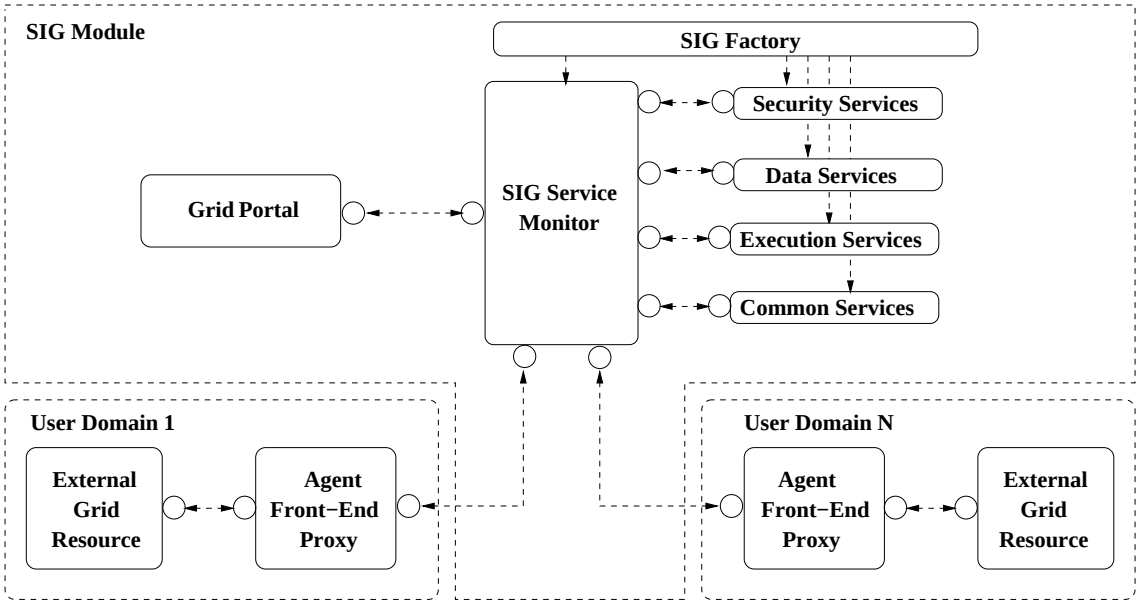


Fig. 1. Modelo para Integração de Recursos em Grade.

A proposta é baseada no modelo do Java CoG Kit mas, diferente dele, fornece uma alternativa mais cômoda para o desenvolvimento de aplicações. Para criar um novo serviço para a Grade, o usuário cria a sua aplicação e a insere na fábrica. O *SIG Agent*, no domínio do cliente, não precisa ser modificado, podendo ser utilizado tanto para acessar os serviços da fábrica quanto para acessar os outros serviços criados pelo usuário. A aplicação do usuário utiliza o *SIG Agent* como uma interface para interagir como conjunto de serviços mapeados para as funcionalidades básicas do GT, tais como: execução de *jobs* (*globusrun*), transferência de arquivos (*gsiftp*), monitoramento de serviços ativos (implementado como um serviço da fábrica), início de proxy (*grid-proxy-init*), finalização de proxy (*grid-proxy-destroy*), entre outros, e para os novos serviços criados pelo usuário. No modelo apresentado foram definidos os seguintes componentes:

- *External Grid Resource*: é o módulo que mantém o conjunto de serviços que controlam o recurso fora do domínio da Grade;
- *Agent Front-End Proxy*: define os parâmetros de acesso ao recurso. Esse módulo atua como uma interface entre a aplicação de controle do recurso e o *proxy* da Grade. Também é responsável por abstrair os comandos do *grid*

middleware nativo do domínio e repassá-los ao serviço solicitado da Grade. Esse módulo permite que recursos sejam adicionados ou removidos sem interferir no funcionamento dos outros recursos;

- *SIG Factory*: permite a oferta de serviços sob demanda. Os serviços de acesso a novos recursos na Grade podem ser instanciados em tempo de execução. Esse módulo instancia os serviços que resolvem a abstração do evento disparado no domínio do usuário. Cada serviço (*Security, Data, Execution, Common*) possui uma interface com os serviços de software do *grid middleware*. Por ser voltado para o GT, esse modelo é em parte similar à proposta deste *grid middleware* porque agrega serviços que devem implementar a interface de acesso às funcionalidades originais do sistema. Os serviços gerados a partir do *SIG Factory* foram:
  a) Módulo *Security Services*: mantém os serviços relacionados ao acesso seguro, tais como validação de *proxy*, emissão de certificados, serviços de delegação, entre outros;
  b) Módulo *Data Services*: contém os serviços que realizam operações de persistência junto ao *grid middleware*;
  c) Módulo *Execution Services*: contém os serviços que se comunicam com as funções de transferência de dados, submissão e escalonamento de tarefas;
  d) Módulo *Common Services*: mantém os serviços que executam serviços específicos de outros ambientes;
  e) *SIG Service Monitor*: define um conjunto de interfaces para os serviços de informação e de monitoramento dos serviços intanciados pelo *SIG Factory*. Esse módulo também monitora os recursos agregados ao domínio. Apesar do MDS (*Monitoring and Discovery Service*) no *Message Service Component* do GT ser capaz de monitorar e descobrir serviços e recursos, o MDS possui dificuldade para descrever recursos não-computacionais [Luo et al. 2009]. O *SIG ServiceMonitor* reconhece que nem toda fonte de informação suporta interfaces *WSRF/WSNotification* e utiliza serviços próprios para resolver essa limitação, mas mantém uma interface com os serviços originais do GT para complementar os serviços originais já oferecidos;
- *Grid Portal*: interface de gerência que permite o acesso seguro do usuário aos recursos agregados.

3.1. Arquitetura para Colaboração em Grade

Os experimentos robóticos são altamente dependentes do desempenho da rede e da precisão dos sistemas de posicionamento e navegação. Por se tratar da interação com agentes móveis, a perda acentuada de pacotes na rede pode levar à acumulação de erros de posição (erros de odometria). Como consequência, o agente será incapaz de atingir o alvo real porque o seu deslocamento no ambiente o levou para uma posição no ambiente diferente da que ele “acredita” ser a posição correta. Isso implica que a infraestrutura deve fornecer um tempo de interação aceitável para atender às requisições do agente.

A arquitetura proposta segue o modelo anteriormente descrito, e é ilustrada na Fig. 2. Para minimizar o esforço para a agregação e interação com os recursos, o *SIG Agent* atua como *middleware* no domínio do usuário e oferece o mapeamento adequado

dos serviços da Grade. Para ampliar a interação com a infraestrutura, o *Grid Proxy* mantém um container de *Web Services* de forma a permitir o acesso de diferentes domínios. A aplicação do usuário pode estar fora dos limites da rede interna da infraestrutura. Por isso, cada usuário precisa ter uma conta na Grade, com os respectivos certificados em seu diretório *home* do *host* de *proxy* (*Grid Proxy*). Os serviços do *SIG Agent* oferecem o mapeamento para o início do *proxy* (*globus-proxy-init*) a partir dos certificados presentes no *Grid Proxy*, e serviços para interação com as demais funcionalidades da Grade. Na arquitetura, os serviços da Grade são mantidos na *SIG Factory*.

O *External Grid Resource* representa o recurso robótico da arquitetura. O *Agent Front-End Proxy* é a interface de acesso, representada pelo *SIG Agent*. Esse componente identifica o recurso, trata o envio e recepção de mensagens e possui a implementação da lógica de acesso ao *Grid Proxy* e ao recurso do domínio do usuário. Dessa forma, o recurso pode ser agregado ao ambiente independente de sua localização, desde que seja fornecido o endereço de *proxy* para o seu domínio. Essa arquitetura suporta o uso de *Web Services* no *Grid Proxy* e no *Grid Portal*. Essa característica é importante porque simplifica o processo de encaminhamento de mensagens entre *firewalls* de domínios distintos, uma vez que são trocadas mensagens XML com o uso do protocolo SOAP (*Simple Object Access Protocol*). O *Grid Proxy* mantém a *SIG Factory* para a instanciação dos serviços de segurança, dados, execução e demais funcionalidades. A arquitetura prevê o uso do GT em todos os nós da rede interna. O *Grid Portal* exibe as informações sobre os recursos que interagem no domínio.

Na arquitetura os *hosts* da Grade mantˆem o relacionamento de confianc¸a com o uso de certificados digitais assinados pela mesma CA. O *Grid Proxy* é o *host* que mantém a fábrica de serviços que abstraem a comunicação com as funções do GT. Portanto, o *Grid Proxy* participa de uma hierarquia na qual tanto o tráfego de ingresso quanto de egresso é filtrado antes de entrar e depois de sair do domínio da Grade, ou seja, apenas são admitidas submissões de *jobs* e o uso de serviços da fábrica por usuários que tenham certificados válidos, dentro do prazo definido na geração do *proxy* para o usuário. Os *nós internos* apenas encaminham as requisições e realizam o roteamento de mensagens, se necessário. O *Grid Portal* é o *host* que mantém a comunicação com os *hosts* de borda do domínio e realiza as funções de gerência e monitoramento de recursos dentro do domínio. Os recursos se comunicam por meio de serviços da fábrica. Esses dados trafegam em mensagens SOAP compactadas que seguem o estilo arquitetural *Representational State Transfer* (REST) [Fielding 2000].

A implementação dessa arquitetura utiliza o GT em *hosts* Linux Dell Quad Core, robôs móveis Pioneer P3-DX e sensores Intel Imote2 com Sistema Operacional TinyOS [TinyOS 2010]. O *SIG Agent* também suporta parcialmente a interação com o *softphone* Ekiga [Ekiga 2010] para estabelecer chamadas SIP (*Session Initiation Protocol*) no domínio do usuário. O *Grid Proxy* executa um servidor Apache Tomcat e um container Apache Axis2 para a comunicação com o *SIG Agent*.

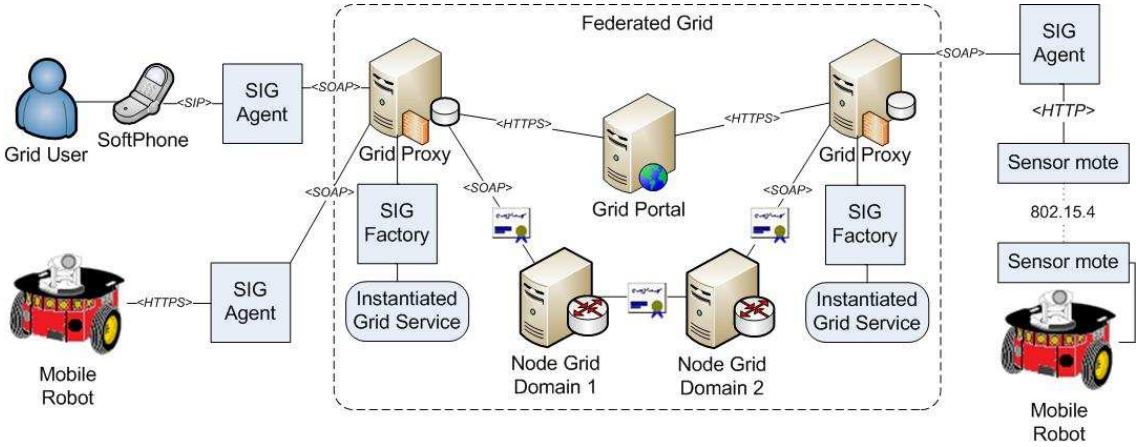


Fig. 2. Arquitetura para Colaboração em Grade.

4. Sistemas Classificadores em Grade

Como exemplo de experimento colaborativo, procurou-se utilizar a Grade como uma infraestrutura capaz de suportar a interação de múltiplos agentes autônomos controlados por Sistemas Classificadores, ativados com serviços da fábrica, que realizam chamadas de sistema às funções mapeadas do GT. Esse estudo considera o uso da Grade como infraestrutura de suporte para experimentos colaborativos.

Sistemas Classificadores foram originalmente propostos por Holland [Holland 1998]. Um Sistema Classificador de Aprendizagem (*Learning Classifier System* - LCS) é um sistema de propósito geral que adota uma metodologia evolucionária para sintetizar mecanismos de inferência adaptativos capazes de operar em condições variadas durante a sua execução [Geyer-Schulz 1995] [Eiben and Smith 2007]. Uma população de regras do tipo *condição-ação-predição*, chamadas classificadores, é mantida para representar a solução atual do problema. Cada classificador mantém uma parte do problema, com a *condição* relacionada às entradas, *ação* relacionada à decisão, e *predição* que estima a relevância do classificador em termos da solução.

O problema da navegação autônoma nos experimentos realizados envolve o alcance de um alvo pré-definido no espaço de busca discretizado. As informações sobre a localização do robô, do alvo e dos parâmetros para a ativação do robô são informados no *job*. No algoritmo descrito a seguir, o método *createRules* gera uma matriz R com classificadores binários aleatórios. Cada linha na matriz R é um classificador com uma estrutura de vetor e o campo *fitness* iniciando com o valor inteiro 100. O método *acquireInfoEnv* calcula a distância que o robô se encontra do alvo, traduz a entrada dos sensores em binário e insere o resultado no vetor C. O método *evaluate* calcula os campos de C que possuem o mesmo valor em cada linha da matriz de classificadores. Para cada combinação positiva, o campo de energia da regra é incrementado, com a conseqüência de gerar classificadores mais específicos para tratar a situação do ambiente. No método *competition* um processo de sinalização é realizado para selecionar os classificadores mais adequados às entradas do ambiente. O campo *status* no classificador recebe o resultado da competição. Inicialmente um processo de previsão é realizado para verificar se o próximo movimento irá orientar o robô móvel em direção

Algorithm 1 Classifier System

Require: population ≥ 1

1: $R \Leftarrow createRules(\#rules)$
2: $C \Leftarrow acquireInfoEnv(sensors)$
3: $e \leftarrow 1$
4: **while** $e \leq epoch$ **do**
5: $g \leftarrow 1$
6: **while** $g \leq generations$ **do**
7: $fit \Leftarrow evaluate(C, R)$
8: $R \Leftarrow competition(R)$
9: $R \Leftarrow action(R)$
10: $R \Leftarrow taxTimeLife(R)$
11: $g \leftarrow g + 1$
12: **end while**
13: $R2 \Leftarrow select(R)$
14: $R2 \Leftarrow reproduces(R2)$
15: $R2 \Leftarrow variate(R2)$
16: $R \Leftarrow renewRules(R2, R)$
17: $e \leftarrow e + 1$
18: **end while**

ao alvo. No experimento, o *fitness* é a relação entre a posição atual do robô e o objetivo a ser alcançado. O *fitness* é calculado como segue:

$$fit = ((1/dis\tan ce) + K) * noise \quad (1)$$

Na Eq.(1), como um problema de minimização, o *fitness* é o inverso da distância Euclidiana ao alvo; *K* é uma constante utilizada para balancear o processo de seleção do tipo *Roulette Wheel* [Eiben and Smith 2007] dos classificadores; *noise* é uma constante estimada fornecida para balancear os erros de deslocamento do robô no ambiente real. Portanto, o *fitness* representa a qualidade do classificador de acordo com as entradas recentes obtidas do ambiente. Um classificador com *fitness* elevado terá maiores chances na fase de competição em razão de estar relacionado a uma menor distância em relação ao alvo. Na fase de competição, cada classificador faz um lance (*Bid*) que depende da especificidade do classificador, energia atual e *fitness*, como descrito a seguir:

$$Bid_t = (spec_t / numSensors) + energy_t + fit_t \quad (2)$$

$$spec_t = (N - total\_\#) / N \quad (3)$$

$$Tax = Bid\_tax * Bid_t \quad (4)$$

onde *t* representa o instante em que os valores foram coletados. Nas Eq.(2) e Eq.(3), o *spec* é uma proporção entre o número total de bits menos os caracteres do tipo *'do not care'* (representados por '#' e que podem representar os bits 0 e 1), em relação ao número total de bits. Na Eq.(2) *numSensors* é o número de sensores do robô; a proporção entre *spec* e *numSensors* é interessante porque mantém o valor de *Bid* bem

distribuído para qualquer número de sensores; o valor *energy* representa a 'força' atual do classificador, ou seja, ao contrário do *fitness* que é válido apenas no movimento atual, o valor de *energy* é mantido nas gerações subsequentes, até que uma *época* seja completada. Na fase de competição uma taxa de participação decrementa o valor de *energy* dos classificadores. Essa atenuação é proporcional ao *Bid* oferecido, com a intenção de evitar valores elevados de *energy*, reduzindo as chances de outros classificadores ganharem nas próximas competições. Na Eq.(4) o valor de *Tax* é a taxa de participação; a contante *Bid tax* é aplicada no valor atual do *Bid* no instante *t*. Depois da fase de competição, a ação no classificador é realizada. De acordo com o resultado dessa ação, a regra é recompensada ou punida. A fórmula geral de cálculo de *energy* a cada *geração* é descrita como segue:

$$E_{t+1} = (1 - life\_tax) * E \ + R_t - Bid\_tax \quad (5)$$

Na Eq.(5), *E* é o valor de *energy* no instante *t*; *life tax* é a taxa de vida que é decrementada do valor de *energy* em cada *geração*; *R* é o valor da recompensa ou punição resultando da ação no instante *t-1*. De acordo com a ação no ambiente, será obtido $R \geq 0$ em uma ação positiva (aproximação ou alcance do alvo), ou $R \leq 0$ caso contrário. Finalmente, *Bid tax* é a constante aplicada ao valor atual de *Bid* no instante *t*. Note que o valor de *Bid* é utilizado para indicar o classificador vencedor na fase de competição. Em cada *geração*, o método *select* recupera o classificador com energia mais elevada, utilizando uma seleção do tipo *Roulette Wheel*, selecionando metade da população de classificadores e armazenando na matriz R2. O método *reproduces* realiza o *crossover* pontual nas regras; *variates* realiza o processo de mutação com inversão pontual de bits, com a intenção de aumentar a diversidade de regras. Finalmente, *renewRules* troca os classificadores com energia mais baixa pelos novos classificadores gerados. Portanto, a competição ocorre entre os descendentes, e um descendente que possui um valor elevado de *fitness* substitui a regra que obteve baixo valor de *energy* em R. Essa é uma estratégia evolucionária do tipo (μ, λ)-EE [Eiben and Smith 2007].

5. Avaliação e Resultados

O Sistema Classificador utiliza os serviços da Grade para realizar operações de telemetria com os sonares do robô (*getSonarReadings*), consulta de posição (*getPosition*), controle de orientação (*setHeading*) e deslocamento (*go*). O algoritmo controla o deslocamento do robô com o envio periódico de requisições, utilizando os serviços da Grade. Essa comunicação fim-a-fim é realizada como mostra o diagrama da Fig. 3. No cálculo do RTT (*Round Trip Time*) , obteve-se um RTT total médio de 950ms (milissegundos) nessa comunicação fim-a-fim.

A convergência do Sistema Classificador foi avaliada com o envio de requisições para o simulador MobileSim [MobileRobots 2010], com uma população de 5 robôs móveis. Na Fig. 4 as linhas espessas representam os caminhos percorridos: (1) o alvo definido em coordenadas cartesianas X=-12000, Y=-12000, escala em milímetros, foi definido para a população; (2) o algoritmo é iniciado e os robôs evitam colidir uns com os outros, mas se deslocam em direção ao alvo; (3) ocorre a convergência do algoritmo e os robôs terminam as suas atividades. A convergência do algoritmo mostra

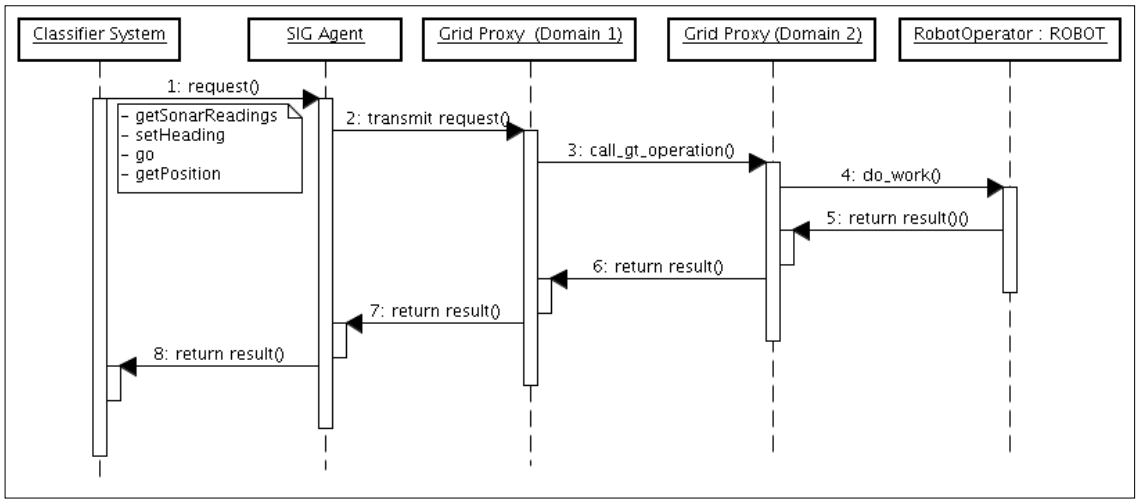
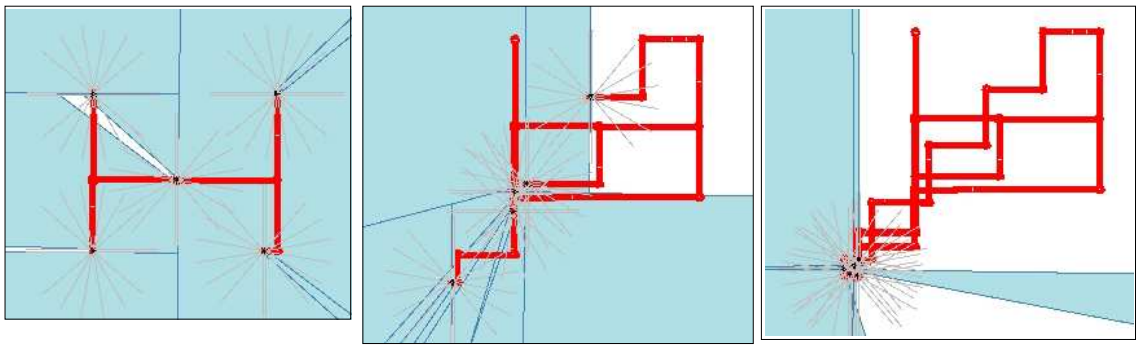


Fig. 3. Interação entre o Sistema Classificador, a Grade e o Robô.



(1) **(2)** **(3)**

Fig. 4. Convergência do Sistema Classificador.

que a infraestrutura é capaz de manter a interação de um conjunto de agentes autônomos, sem afetar o resultado da execução do algoritmo.

Outra classe importante de recursos da Grade são pequenos dispositivos sensores interligados em rede, como ilustrado na Fig. 2. Apresentamos a seguir um experimento envolvendo o acesso por meio da Grade a sensores Imote2. Um dos sensores tem a função de *gateway* interligando a Grade aos demais sensores da rede, um dos quais embarcado no robô. Nesse experimento foram realizados testes de RTT com um cliente diretamente conectado a um *Grid Proxy* e que consulta o recurso Mote que está conectado a outro *Grid Proxy*, de acordo com a Fig. 2. A Fig. 5 mostra o RTT total da arquitetura (RTT1): acesso ao *Grid Proxy*, transferência de dados na rede interna, acesso ao Mote, e retorno da resposta para o cliente. A comparação é feita com o RTT do acesso interno ao Mote (RTT2), a partir do *Grid Proxy* ao qual ele está conectado. Os resultados obtidos mostram que o desempenho da arquitetura é suficiente para o uso de aplicações que exigem reduzida latência para a execução de suas atividades. Essa característica é importante para diversos algoritmos, como os de computação evolutiva, que exigem a interação contínua com os serviços de correção de odometria para a navegação autônoma.

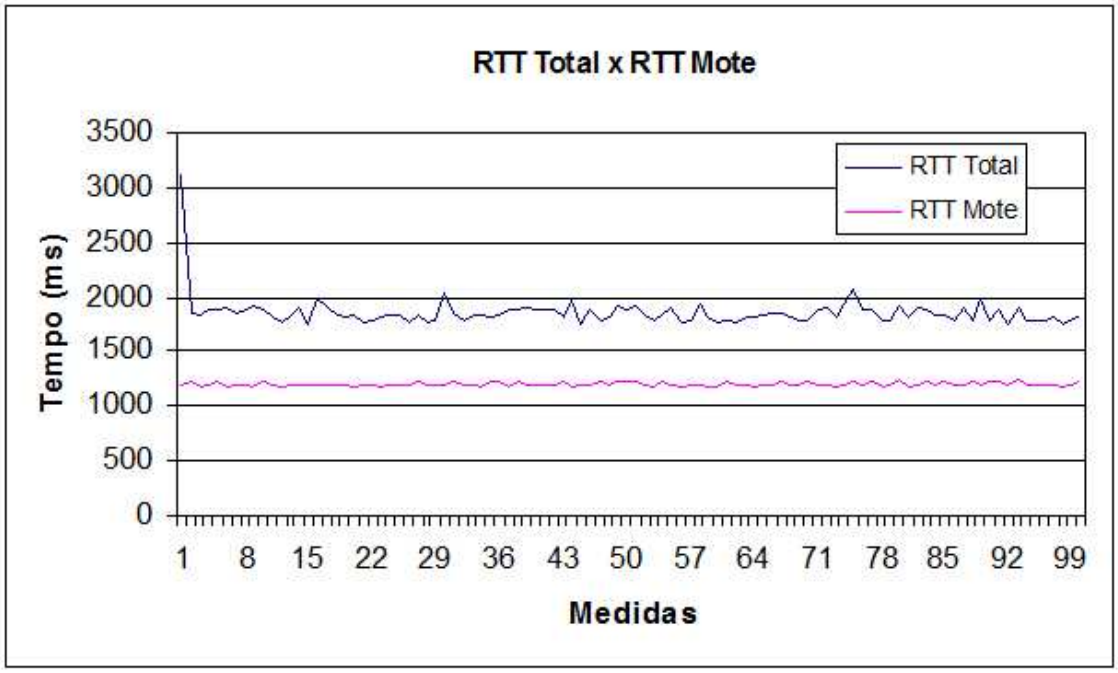


Fig. 5. Avaliação do Desempenho da Arquitetura.

A Tab. 1 mostra os resultados obtidos para uma população de 100 amostras para o RTT1 e RTT2. São apresentados também o desvio-padrão (DP) e o intervalo de confiança (IC) para um nível de significância de 0,05. Os resultados mostram que o desempenho da arquitetura permite a realização de experimentos remotos em um tempo aceitável e inferior, em média, a 1 segundo (diferença entre RTT1 e RTT2). Essas avaliações são interessantes porque permitem definir nos experimentos práticos o tempo de espera entre as requisições a recursos de baixo processamento, sem a perda acentuada de pacotes. No caso do recurso Mote, o tempo de espera entre as requisições do cliente deve ser superior à média do RTT1. Essa mesma métrica pode ser utilizada sem perda de generalidade para o acesso a outros recursos, sem considerar a sobrecarga do uso compartilhado da rede.

Média do RTT1 (ms)	DP	IC	Média do RTT2 (ms)	DP	IC
1858,14	143,309	28,088	1206,66	14,644	2,870

Tab. 1. Comparação entre RTT1 e RTT2.

6. Considerações Finais e Trabalhos Futuros

A avaliação inicial da arquitetura mostra que uma grande diversidade de experimentos pode ser realizada no ambiente. O suporte à agregação dinâmica de recursos simplifica o uso desse ambiente respeitando os quesitos de segurança. A arquitetura é uma alternativa para que experimentos colaborativos possam ser realizados com a abstração da lógica de configuração do domínio. Como trabalho futuros pretende-se estender o conjunto de experimentos e utilizar uma interface de interação colaborativa junto a um *workflow*, para submeter tarefas que exigem o processamento paralelo.

Referências

Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. (2003). *"OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing"*. Number ISBN 978-3-540-20405-3. Springer Berlin / Heidelberg.

Breitling, F., Granzer, T., and Enke, H. (2008). "*Grid Integration of Robotic Telescopes*". *Astronomische Nachrichten*.

Cazangi, R. R. (2008). *"Síntese de Controladores Autônomos em Robótica Móvel por meio de Computação Bio-inspirada"*. PhD thesis, Universidade Estadual de Campinas.

Cirne,W. and Neto, E. S. (2005). "*Grids Computacionais: Transformando a Computação através de Serviços sob Demanda*". *XXIII SBRC*.

Eiben, A. E. and Smith, J. E. (2007). *"Introduction to Evolutionary Computing"*. Springer, Natural Computing Series.

Ekiga (2010). Ekiga - free your speech. URL at `http://ekiga.org`.

Fielding, R. T. (2000). *"Architectural Styles and the Design of Network-based Software Architectures"*. PhD thesis, University of California.

Foster, I. and Iamnitchi, A. (2003). "*On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing*". Pages 118–128.

Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). "*The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*".

Foster, I. and Tuecke, S. (2005). "*The Different Faces of IT as Service*". Enterprise Distributed Computing - ACM Queue, 3(6).

Geyer-Schulz, A. (1995). "*Holland Classifier Systems*". SIGAPL APL Quote Quad Journal, 25(4):43–55. ISSN 0163-6006.

GGF (2010). Open Grid Forum. URL at `http://www.ggf.org`.

Globus (2010). The Globus Alliance. URL at `http://www.globus.org`.

GRID Infoware (2010). Grid Computing Info Centre (Grid Infoware). URL at `http://www.gridcomputing.com`.

Guillen, X. V. (2009). *"Distributed Resource Allocation for Contributory Systems"*. PhD thesis, Universitat Oberta de Catalunya.

Holland, J. H. (1998). "*Emergence: from chaos to order*". Addison Wesley, Inc.

IT Professional (2004). "*Grid Computing 101: What's All the Fuss About?*". IT Professional Magazine.

Krishnaswamy, R., Navarro, L., and Vlassov, V. (2008). "*A democratic grid: Collaboration, sharing and computing for everyone*". In eChallenges'08 Collaboration and the Knowledge Economy: Issues, Applications, Case Studies, Stockholm. IOS Press.

Laszewski, G. v., Hategan, M., and D., K. (2010). "*Java CoG Kit*". URL at `http://wiki.cogkit.org/wiki/Java CoG Kit`.

Luo, S., Peng, X., Fan, S., and Zhang, P. (2009). "*Study on Computing Grid Distributed Middleware and Its Application"*. International Forum on Informtion Technology and Applications.

MobileRobots (2010). Mobile Robots Inc. URL at `http://www.mobilerobots.com`.

OASIS (2010). Advancing Open Standards for the Information Society. URL at `http://www.oasis-open.org`.

OGF (2010). Open Grid Forum. URL at `http://www.ogf.org/UnderstandingGrids/resource_articles.php`.

Parker, P. M. (2010). "*Webster's Online Dictionary"*. URL at `http://www.websters-online-dictionary.org`.

Peterson, L., Anderson, T., Culler, D., and Roscoe, T. (2006). "*Experiences Building PlanetLab*". Proceedings of the Seventh Symposium on Operating System Design and Implementation (OSDI). URL at `http://www.planet-lab.org/about`.

Reis, L. P. (2003). "*Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico"*. PhD thesis, FEUP.

Sabatier, F., De Vivo, A., and Vialle, S. (2004). "*Grid Programming for Distributed Remote Robot Control*". Pages 358 – 363.

Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C., and Casanova, H. (2002). *"Overview of GridRPC: A Remote Procedure Call API for Grid Computing"*, volume 2536/2002. Springer Berlin / Heidelberg.

Siegwart, R. and Nourbacksh, I. R. (2004). *"Introduction to AutonomousMobile Robots"*. The MIT Press.

Sotomayor, B. (2005). *"The GT 4 Programmer's Tutorial"*. Apache Public License.

TinyOS (2010). Tinyos Community Forum. URL at `http://www.tinyos.net`.

Vilajosana, X., Daradoumis, T., Navarro, L., and Manuel, J. (2007). "*LaCOLLA: Middleware for Self-Sufficient Online Collaboration*". IEEE Internet Computing, 11:56–64.

Ziviani, A. and Duarte, O. C. M. B. (2005). "*Metrologia na Internet*". *XXIII SBRC*.

Previsão de Utilização de Recursos por Aplicações no InteGrade

Fábio Augusto Firmo[1], **Marcelo Finger**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo
Rua do Matão, 1010 – Butantã – 05.508-090 – São Paulo – SP – Brasil

fafirmo@gmail.com, mfinger@ime.usp.br

Abstract. *In this paper we present a method to obtain predictions on the amount of computational resource used for a given distributed grid application. The resources focused are RAM memory, CPU use and execution time. For that we use several low-cost techniques that allow for an easy implementation and low overhead. To demonstrate its feasibility, we compared them against more sophisticated models from other authors; in addition, we offer an implementation integrated with the InteGrade middleware.*

Resumo. *Apresentamos neste trabalho um método para obter previsões da utilização de recursos computacionais, como memória, processamento e tempo de execução, que serão utilizados por uma aplicação em uma grade oportunista. Para isso fazemos uso de técnicas simples, escolhidas visando uma fácil implementação e baixa sobrecarga. Para atestar sua viabilidade, comparamos estas técnicas com métodos mais sofisticados encontrados na literatura e disponibilizamos uma implementação de nossas técnicas integrada ao middleware InteGrade.*

1. Introdução

O InteGrade [sit 2010] é um *middleware* para grades oportunistas desenvolvido por diversas instituições pelo Brasil. Seu objetivo é aproveitar os recursos de uma grade computacional para executar aplicações computacionalmente complexas. Uma grade computacional consiste em uma rede de computadores que compartilham seus recursos sob demanda para realizar uma determinada tarefa. Grades oportunistas utilizam tipicamente máquinas compartilhadas, como laboratórios de organizações e empresas, e apenas utilizam seus recursos ociosos.

Em ambientes como esse, escalonar um trabalho para um nó que será ocupado por seu usuário antes do término da computação compromete a Qualidade de Serviço oferecida ao usuário, pois apesar de executar suas aplicações em prioridade mínima, o InteGrade ainda não é capaz de limitar a utilização de recursos tomados. Além disso a vazão da grade diminui, pois o trabalho executa mais lentamente ou é cancelado. Esse cenário estimula o desenvolvimento de mecanismos que diminuam a ocorrência de tais eventos.

Uma possibilidade é prever os períodos de disponibilidade das máquinas. Para isso foi criado um módulo, chamado *Local Usage Pattern Analyzer* (LUPA) [Bezerra 2006, Conde 2008, Finger et al. 2009], capaz de prever por quanto tempo um nó pode oferecer uma determinada quantidade de recursos. Entretanto é necessário que o dono da aplicação

forneça uma estimativa de tempo, processamento e memória necessários para que sua aplicação termine. Essa não é uma boa abordagem pois essas informações são geralmente desconhecidas, além de tornar a experiência de submissão mais complexa e demorada. Devido a esses fatores a previsão fornecida ao LUPA é geralmente muito imprecisa ou nem mesmo fornecida, levando ao subaproveitamento da funcionalidade.

Nosso objetivo é construir um novo módulo do InteGrade para que o LUPA funcione sem qualquer entrada do usuário. Isso significa produzir uma expressão que contenha a taxa de utilização de CPU, memória e tempo mínimos para que uma aplicação termine.

Embora em um primeiro momento nos preocupamos em estudar e simular separadamente vários métodos para resolver o problema, é importante que exista uma implementação pronta e bem integrada ao projeto, e não somente resultados teóricos. É necessário também que essa implementação seja simples, de modo a minimizar o impacto que o cálculo das previsões causará nas máquinas da grade. Optamos por começar com métodos mais simples quanto possível e continuar com um processo de refinamento até encontrar um método que forneça uma previsão com uma precisão e sobrecarga aceitáveis em termos práticos.

Não está no escopo deste trabalho qualquer atividade relacionada diretamente ao escalonamento das aplicações. Essa é uma área que demandaria muito mais experimentos para obter alguma conclusão. A idéia é fornecer boas estimativas para o LUPA, que assim poderá fornecer boas estimativas ao escalonador.

O artigo está organizado da seguinte maneira: a segunda seção contém uma breve descrição dos trabalhos já publicados sobre o assunto. A seção 3 apresenta uma descrição detalhada da abordagem escolhida. As simulações e seus resultados são apresentados na seção 4. A seção 5 dedica-se a explicar sucintamente a implementação do método no InteGrade e por fim as conclusões são apresentadas na sexta e última seção.

2. Trabalhos relacionados

Uma das abordagens estudadas tenta modelar o comportamento da aplicação ao longo de sua execução [Dodonov et al. 2007, de Mello and Yang 2009]. Várias métricas, como utilização de CPU, rede, requisições de disco e mensagens MPI, são monitoradas a fim de encontrar padrões de comportamento. A partir dessa classificação é possível prever, utilizando técnicas variadas como Cadeias de Markov e teoria do caos, quando e para qual padrão a aplicação se comportará em seguida. Isso permite que o gerenciador da grade tome medidas ao longo da execução, como por exemplo migrar o trabalho para um nó mais indicado para o futuro estado.

Outros pesquisadores desenvolveram sistemas capazes de personalizar suas previsões para aplicações específicas, como por exemplo o Cactus [Liu et al. 2002], utilizado em pesquisa astrofísica.

Grande parte da literatura estudada, entretanto, concentra-se em realizar suas estimativas analisando observações passadas. A essência das abordagens é a mesma: é preciso definir um subconjunto de execuções passadas que se assemelhe à recém-submetida. Em seguida as características desse subconjunto são processadas para obter a previsão desejada. As diferenças geralmente estão nas definições de similaridades, nos mecanismos de busca de objetos similares e nas técnicas de previsão aplicadas às execuções passadas.

Smith, Foster e Taylor [Smith et al. 1998] agruparam trabalhos com certas características idênticas em *templates*. Para definir quais são as características relevantes em cada submissão foram testados um algoritmo guloso e um genético. Finalmente, foi levado em conta a média ou a regressão linear do subconjunto gerado pelo *template*. Essa abordagem foi generalizada para um modelo baseado em *Case-based Reasoning* (CBR). A principal diferença é que a comparação de características deixa de ser binária, dando lugar a funções de distância, permitindo assim selecionar os pontos mais próximos para a análise.

Em [Nassif et al. 2007] os autores utilizaram CBR para construir um sistema distribuído para grades. Um método mais eficiente para a busca dos vizinhos mais próximos, chamado *refined nearest neighbour algorithm* foi utilizado, apresentando bons resultados. Já em [Li 2007], o autor utilizou uma estrutura de dados mais complexa (árvore-M) para resolver o problema de desempenho da busca. Além disso são propostos algoritmos para ajuste automático de parâmetros de busca e um sistema adaptativo que pode incorporar informações da política de escalonamento quando conveniente.

3. Previsão de Recursos de Aplicações

Com base na análise da literatura pudemos definir algumas características desejáveis do nosso modelo. Um conceito fundamental desse trabalho é a simplicidade. Embora o problema seja complexo optamos por testar primeiramente as opções mais simples, testar sua eficácia, e refiná-las se necessário. Uma grande razão para essa política está na implementação, pois um modelo mais simples torna a implementação mais rápida, o código mais estável, a manutenção mais fácil e reduz o impacto no desempenho das máquinas da grade.

Outra decisão foi em relação à granularidade da análise da uma execução. Nesse contexto, chamamos de **análise local** a análise e previsão de utilização de recursos durante a execução da aplicação. Nesse caso a utilização de recursos da aplicação é resumida em uma série temporal. Em oposição, a **análise global** desconsidera em que ponto da execução os eventos ocorreram, interessando-se apenas por números representativos dessa utilização, como a média, mínimo ou máximo. Escolhemos a abordagem global por dois motivos: por sua simplicidade, e pelo pouco valor em saber em que período da execução os recursos serão utilizados, já que idealmente o nó estará ocioso a maior parte do tempo e pela dificuldade na migração de trabalhos dentro da grade.

3.1. Definindo similaridade

Nesse trabalho usamos observações passadas para encontrar uma boa previsão para o próximo trabalho. Analisar todas as execuções já realizadas não é uma boa idéia. É preciso definir um subconjunto de trabalhos com características similares às da nova aplicação. Esse é um típico problema de Aprendizado Computacional, e nessa seção apresentaremos algumas soluções propostas por outros autores, assim como os mecanismos que utilizamos em nosso modelo.

Uma das primeiras preocupações é definir quais informações são relevantes para a comparação. A lista de características (ou atributos) de cada execução é extensa. Algumas das mais intuitivas são: nome da aplicação, nome do usuário, argumentos do executável, tamanho dos arquivos de entrada e número de nós. Porém há também outras menos

evidentes, como horário da submissão, "idade" e tamanho do executável; e ainda aquelas particulares da grade ou *cluster* em questão, como tipo de prioridade (ou tipo de fila) e grupos de usuários.

Além disso é necessário definir, para cada característica, uma função que permita comparar duas execuções. Chamamos de **função de similaridade** uma função sim que mapeia duas características ao intervalo $[0, 1]$. $sim(a, b) = 1$ se e somente se a e b são consideradas iguais.

Essa escolha não é trivial. Tome o exemplo do argumento do executável, uma possibilidade é utilizar uma função binária, ou seja, caso as duas cadeias de caracteres sejam exatamente iguais, a similaridade é um, e zero caso contrário. Isso implica que, por exemplo, os argumentos `--width=10 --height=20` e `--height=20 --width=10` serão considerados diferentes, embora provavelmente deveriam ser tratados como iguais.

Com isso podemos calcular a similaridade local entre dois pontos, ou seja, mensurar a semelhança de suas características. O próximo passo é encontrar a similaridade global, que é a semelhança entre os duas execuções. Para isso basta atribuir pesos aos atributos presentes no modelo. Isso pode ser representado por:

$$sim(A, B) = \sum_{i=1}^{c} \omega_i sim_i(a_i, b_i)$$

onde A e B são execuções, c é o número de características e sim_i é a similaridade local para a característica i.

A última etapa consiste em delimitar o subconjunto similar ao objeto de previsão. Uma alternativa comum é escolher um limiar adequado e encontrar todas as execuções passadas com similaridade maior ou igual. Outra possibilidade é encontrar um número adequado de execuções mais próximas ao ponto de consulta, conhecida como busca de vizinhos mais próximos.

A utilização do algoritmo dos vizinhos mais próximos acaba levando a um possível problema de desempenho, pois o tempo de busca pode se mostrar muito grande à medida que a base de dados cresce. Esse fato, além de atrasar a submissão da nova aplicação, pode representar um pico de processamento indesejado e, conseqüentemente, queda na Qualidade de Serviço, pois muitas vezes a busca será executada em um nó não dedicado. Além disso foi constatado empiricamente [Smith et al. 1998] que, assim como é possível deduzir intuitivamente, o nome da aplicação é o atributo mais relevante, com uma grande margem em relação aos demais. Esses dois motivos nos levaram a adotar uma definição de similaridade extremamente simples, considerando como similares todas as execuções realizadas da mesma aplicação.

3.2. Métodos de previsão

Assim que encontramos um conjunto de aplicações similares ao novo caso, o próximo passo é calcular a estimativa. Para isso utilizamos algumas estatísticas simples, descritas a seguir:

- **Média**
  A média aritmética, apesar de bastante simples, foi utilizada várias vezes com

bons resultados em diversos trabalhos anteriores. Além disso possui a grande vantagem de ser calculada de maneira bastante rápida.

- **Máximo**
  Utilizar o valor máximo das execuções passadas, ou o máximo mais o desvio padrão, é uma maneira garantida de conseguir estimativas conservadoras, que podem ser interessantes dependendo da política de escalonamento. Essa seria uma opção razoável quando a aplicação não apresentasse nenhuma execução anormal que eleve muito as próximas previsões, levando a grandes erros continuamente. Todas as aplicações encontradas em cargas de trabalho reais, contudo, apresentaram picos, o que inviabilizou a adoção desse método.
- **Máximo com decaimento**
  O maior problema em utilizar o máximo é fixar todas as próximas previsões para valores maiores ou iguais a ele. Isso implica que o erro causado por um ponto extremo é propagado indefinidamente. Há algumas maneiras de evitar isso, como por exemplo considerar apenas um número fixo das execuções mais recentes. Optamos por multiplicar a previsão por uma função decrescente, no caso e^{-x}, onde x é o número de previsões feitas desde o último erro. Isso significa que a cada vez que uma previsão se mostra insuficiente, ou muito próxima ao valor observado, a próxima previsão voltará ao valor máximo.
- **Intervalo de confiança**
  Pode ser considerado como um refinamento da média, pois também leva em consideração o espalhamento dos dados. Como se trata de um intervalo consideramos o valor do limitante superior. Em todas as simulações utilizamos intervalos com 95% de confiança.
- **Mediana**
  Outra método simples. É interessante por ser menos suscetível a pontos extremos que a média e o intervalo de confiança.
- **Regressão Linear**
  Não foi utilizado nas simulações por apresentar menor precisão que a média em [Smith et al. 1998].

Um detalhe importante é que consideramos todas as execuções passadas com o mesmo peso, porém esse não é a única opção. Poderia ser possível adotar um peso à cada execução baseado em sua "idade" ou em sua distância ao ponto a ser estimado.

3.3. Tipos de recursos

O modelo apresentado não faz nenhuma suposição sobre que tipo de recurso será analisado, porém é preciso tomar alguns cuidados nesse quesito.

Uma limitação é causada pelo LUPA. Atualmente esse módulo só é capaz de analisar padrões de utilização de CPU e memória, portanto não faz sentido realizar previsões sobre outros recursos, como utilização de disco, por exemplo, embora seja possível estender os dois sistemas para isso.

Outro detalhe importante é que nem todos os recursos apresentarão necessariamente o mesmo comportamento. Nada garante que um bom modelo de previsão para o tempo de execução apresente mesmo desempenho para memória. O mesmo vale para aplicações diferentes. Pensando nisso tomamos cuidado para não fixar nenhum método de

previsão, tornando possível que cada aplicação e cada recurso tenha seu próprio método, com parâmetros possivelmente personalizados.

Por fim, até agora consideramos o tempo de execução apenas como seu tempo absoluto em segundos porque, entre outros motivos, a maioria dos trabalhos realizou suas simulações desse modo, tornando a comparação mais fácil. Porém no cenário real é preciso levar em consideração a heterogeneidade da grade e a carga de CPU nos nós. Para isso foi necessário definir uma medida da capacidade de processamento, que poderia ser utilizada para normalizar o tempo de execução. A medida escolhida foi o valor de *BogoMips* (*Bogus million instructions per second*) do processador que, apesar de suas deficiências, mostrou uma precisão satisfatória, além de ser compatível com a filosofia de simplicidade adotada em todo o trabalho.

4. Simulações

Durante todo o processo de planejamento realizamos várias simulações a fim de obter informações sobre a viabilidade do modelo proposto antes de iniciar a implementação no InteGrade.

A maioria das simulações feitas, e todas apresentadas nesse artigo, foram realizadas com base em cargas de trabalho reais de *clusters* obtidas através do Parallel Workload Archive [wor]. Escolhemos cargas de trabalho que apresentam, para cada execução, o tempo de execução e o máximo de memória requisitada. Os dois registros utilizados são descritos na Tabela 1, que também indica quantas execuções foram de fato analisadas, pois encontramos muitos casos em que uma aplicação era executada apenas algumas vezes durante todo o período. Para as simulações descritas nesse artigo apenas consideramos aplicações que foram executadas mais que 20 vezes.

Tabela 1. Cargas de trabalho utilizadas

Nome	Período	Execuções analisadas	Execuções totais
LANL	Outubro/1994 até Setembro/1996	4.431	201.387
DAS2-4	Janeiro/2003 até Dezembro/2003	17.447	33.795

Várias execuções foram descartadas em limpezas sugeridas pelo próprio administrador dos *logs*, outras não puderam ser utilizadas porque pertenciam a aplicações que não foram executadas em quantidade suficiente para formar uma base razoável para realizar as estimativas, tipicamente menos que 30 vezes.

Embora seja possível identificar execuções da mesma aplicação, não é possível obter mais informações sobre a aplicação, como linguagem em que foi escrita, argumentos da execução ou qual o problema que ela resolve. Não sabemos assim, por exemplo, se determinada aplicação realiza muitas operações de ponto flutuante ou muitas consultas em um banco de dados. Como a abordagem é genérica e não faz nenhuma suposição sobre as aplicações, essa falta de informação não compromete os resultados das simulações, porém elas poderiam ser interessantes para uma análise menos especulativa dos resultados e uma melhor segurança para propostas de refinamento.

As simulações foram realizadas da seguinte forma: para cada execução de uma determinada aplicação, calcula-se uma previsão do recurso baseada estritamente nas in-

formações passadas, que então é comparada com o valor real. Execuções que não terminaram com sucesso ou que terminaram em menos de 15 segundos foram ignoradas.

4.1. Métricas de precisão

Utilizamos diferentes métricas para mensurar a precisão das previsões, descritas a seguir:

- **Erro:** Valor absoluto da diferença entre o recurso estimado e o recurso observado.
- **Erro relativo:** É a razão entre a média do erro e a média do recurso observado. Um erro relativo de 0,5, por exemplo, diz que as previsões ficaram, na média, 50% maiores ou menores que o valor observado. Pode-se dizer que é a métrica mais "geral" para comparação de dois métodos, e é utilizada pela maioria dos trabalhos relacionados.
- **Desperdício:** Proporção das previsões que foram maiores que o recurso observado. Nesses casos é provável que algumas máquinas que poderiam ceder recursos não foram cogitadas pelo escalonador devido à estimativa demasiada. No entanto, como o InteGrade não faz nenhum tipo de reserva de recursos, ao término da aplicação o nó estará apto a receber outra aplicação normalmente.
- **Estimativa insuficiente:** Proporção das previsões que foram menores que o recurso observado. Nesses casos é possível que a aplicação exceda o limite "garantido" pelo LUPA, o que pode resultar, por exemplo que a aplicação ainda não tenha terminado quando a máquina é retomada pelo usuário local.

Devido à política do InteGrade em priorizar os donos dos nós compartilhados, consideramos estimativas insuficientes mais graves que desperdícios.

4.2. Resultados

Ao analisar visualmente os gráficos gerados pelas simulações, foi possível encontrar algumas tendências nos comportamentos das aplicações analisadas. O tempo de execução na maioria das simulações manteve-se estável em boa parte do tempo, com picos ocasionais. Esse comportamento é ilustrado pela Figura 1, que mostra para a aplicação de número 3 da carga de trabalho DAS2-4 o tempo de execução e a previsão gerada pelo método de intervalo de confiança.

A previsão de memória mostrou-se um pouco mais instável do que o tempo de execução, porém com menor ocorrência de grandes picos. Essa instabilidade, porém, não prejudicou a previsão, que apresentou erro menor que a de tempo de execução.

Tabela 2. Média dos erros relativos do tempo de execução e memória

	Tempo de execução		Memória	
Cluster	Intervalo de Confiança	Mediana	Intervalo de Confiança	Mediana
LANL	1,18	0,86	0,7	0,64
DAS2-4	1,08	0,75	0,67	0,58

Outra característica importante mostrada nos experimentos é a proporção de estimativas insuficientes em relação ao total de previsões. A Figura 2 mostra os dois métodos aplicados a uma mesma aplicação, onde é possível perceber que essas estimativas indesejadas são mais freqüentes utilizando a mediana, apesar do erro relativo menor.

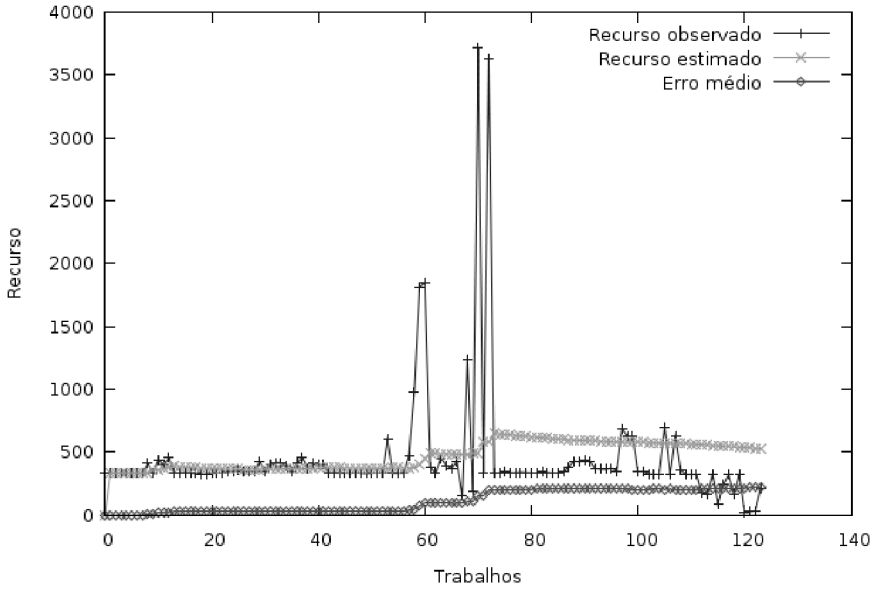


Figura 1. Previsão do tempo de execução de uma aplicação utilizando intervalo de confiança

Esse mesmo gráfico também aponta um outro resultado: o erro médio é consideravelmente menor quando a estimativa é mais alta que o necessário. Essa informação se torna mais nítida ao relembrar os picos mencionados anteriormente. Quando a aplicação se mantém estável, o erro da previsão tende a ser pequeno, porém quando ela demonstra algum pico de utilização de recurso o erro é muito maior. Esse fato aponta que o erro das previsões na verdade é menor que sua média na maioria dos casos, porém é elevado muito pelo erro causados por tais picos, de natureza mais instável e imprevisível.

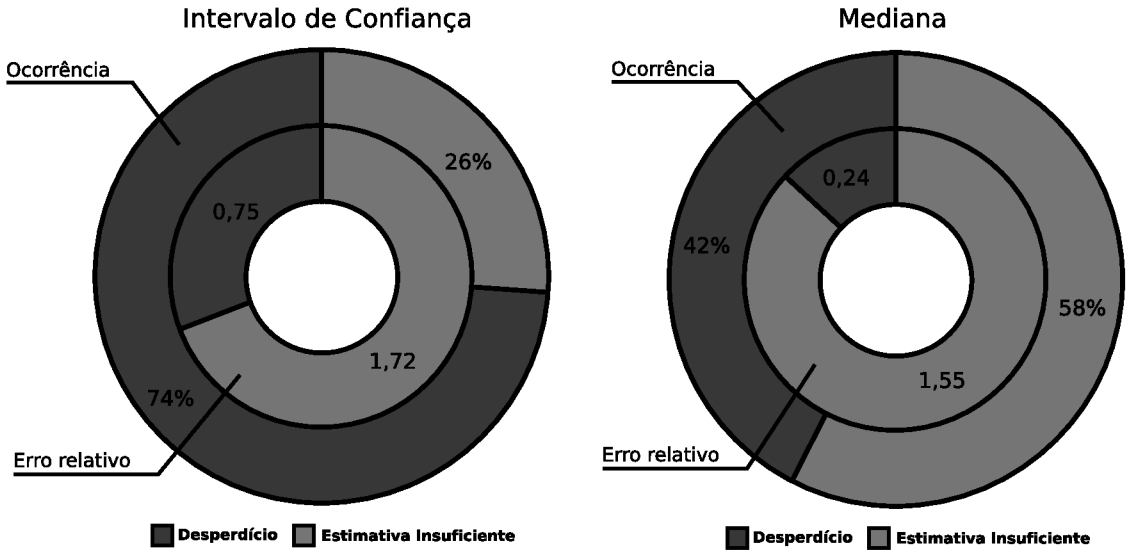


Figura 2. Proporção de desperdícios e estimativas insuficientes

4.3. Comparação com outros trabalhos

Outros autores também utilizaram simulações para mensurar a precisão de maneira muito similar ao que fizemos, possibilitando assim uma comparação. Os resultados do modelo referenciado nesse artigo por Li foram retirados de [Li 2007], já os resultados restantes foram encontrados em [Smith et al. 1998], que apresenta uma comparação do modelo proposto pelos autores Smith, Foster e Taylor com modelos de Gibbons e Downey.

Utilizamos apenas o erro relativo como métrica, pois essa era a única em comum. Para todos os trabalhos, calculamos o erro relativo médio e o erro relativo no melhor caso e no pior caso. Esses dados são mostrados na Tabela 3 e na Figura 3.

Tabela 3. Comparação do erro relativo com outros trabalhos da literatura

	Erro relativo médio	Erro relativo mínimo	Erro relativo máximo
Intervalo de Confiança	1,12	0,50	1,47
Mediana	0,80	0,33	0,93
Smith, Foster, Taylor	0,49	0,39	0,58
Gibbons	0,71	0,68	0,77
Downey (Mediana)	0,88	0,58	0,99
Downey (Média)	1,46	0,61	2,04
Li	0,52	0,49	0,58

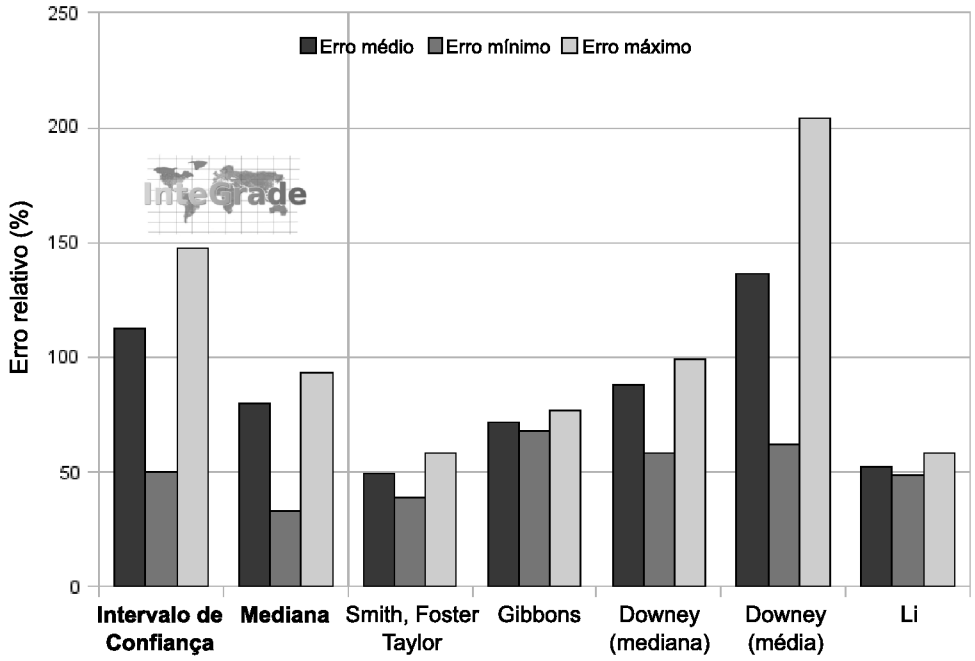


Figura 3. Gráfico comparativo com outros trabalhos da literatura

Considerando apenas o erro relativo, as previsões feitas com a mediana apresentaram melhores resultados. Isso se deve ao fato da mediana ser menos sensível a valores extremos, que foram constatados em todas as aplicações observadas até agora. Outras métricas, entretanto, fornecem informações adicionais sobre os métodos. Em particular,

foi possível notar que a quantidade de estimativas insuficientes é menor ao se utilizar intervalo de confiança. Por fim, constatamos que o erro médio apresentado pelo nosso modelo não está muito longe do erro de trabalhos similares.

As simulações também mostraram que a previsão do tempo de execução e da memória apresentaram resultados similares. Isso indica que o modelo pode ser utilizado para prever diferentes recursos, embora não descartamos a possibilidade de pesquisar maneiras de explorar possíveis particularidades de recursos específicos.

5. Implementação

Era imprescindível que houvesse uma implementação em produção dessa nova funcionalidade. Essa capítulo descreve a arquitetura do InteGrade e das modificações necessárias, bem como as intenções de possíveis melhorias. Todo o código pode ser encontrado no *branch* resourcePredictor no repositório público do projeto [rep]. Descrições mais detalhadas sobre o InteGrade podem ser encontradas em [Goldchleger 2004] e [Goldchleger et al. 2004]

5.1. Arquitetura do InteGrade

O InteGrade é estruturado como uma hierarquia de aglomerados, ou *clusters*. Um aglomerado é um conjunto de máquinas, geralmente em um mesmo espaço físico ou organização, com um nó gerenciador. Todos os nós gerenciadores são organizados em uma árvore que forma toda a grade.

Cada nó de um aglomerado é classificado em relação à sua função. O **gerenciador do aglomerado** é responsável por gerenciar o seu funcionamento, escalonar as aplicações recebidas e se comunicar com outros aglomerados. O **nó de usuário** é responsável por oferecer ao usuário da grade a interface de submissão de aplicações e exibir seus resultados. Finalmente o **provedor de recursos** é o nó onde as aplicações são executadas. Ele pode ser **dedicado**, caso ceda todo seu potencial computacional à grade, mas tipicamente é **compartilhado**, quando só é aproveitado nos períodos em que o usuário local não o utiliza. É importante notar que essas categorias não são mutuamente exclusivas, um provedor de recursos pode também ser um nó de usuário, por exemplo.

5.2. Funcionamento do LUPA

Localizado nos provedores de recursos, o LUPA é o módulo responsável por analisar os padrões de utilização dos nós da grade e responder a consultas sobre sua futura disponibilidade. Ele utiliza técnicas de *clustering* para os períodos em que a utilização dos recursos são semelhantes. Os recursos são coletados a cada cinco minutos e armazenados em um arquivo de *log* e, uma vez por dia, os *clusters* são atualizados. Assim é possível identificar, por exemplo, que uma máquina é geralmente utilizada em dias de semana pela manhã e tarde, porém permanece ociosa à noite e nos finais de semana.

Essas informações são armazenadas nas próprias máquinas, pois é razoável supor que muitos usuários não desejem que todo o histórico de utilização de seus computadores seja transmitido pela rede e armazenado em outro nó. Toda a comunicação do LUPA é feita através de consultas feitas pelo gerenciador do aglomerado.

As consultas, chamadas no InteGrade de *constraints*, são feitas em linguagem TCL [tcl]. Elas podem conter um limite de utilização de CPU, memória e tempo como,

por exemplo `cpuUsage < 60 and freeRam > 10000 and hours == 2`. O valor de `cpuUsage` deve estar entre 0 e 100, e representa a taxa média de CPU livre. `freeRam` deve ser fornecido como o valor absoluto em *kilobytes*. Antigamente o valor de `hours` deveria ser um valor inteiro, porém isso prejudicaria demais as previsões. Devido a essa limitação modificamos o LUPA para trabalhar também com valores não inteiros no campo horas.

5.3. Arquitetura da implementação

As mudanças ocorreram em diversos pontos do código, porém podem ser resumidas na criação de três novos pacotes:

ApplicationHistoryDatabase Tem como finalidade armazenar de maneira persistente as informações sobre todas as execuções passadas. É uma base de dados centralizada, ou seja, só há uma por aglomerado. Isso evita que todos os nós enviem pela rede as informações sobre as execuções anteriores a cada vez que uma nova previsão precisa ser feita. Também evita a volatilidade de informações, pois provedores de recursos antigos podem ser removidos momentaneamente ou permanentemente. A comunicação dessa nova classe é feita pela extensão da interface CORBA do InteGrade.

ResourcePredictor Agrupa as classes relativas à previsão dos recursos. Cada método de previsão, como mediana ou intervalo de confiança, é representado por uma classe que implementa a interface `ResourcePredictor`. Essa interface contém apenas um método, o `predictNext`, que recebe um nome de aplicação e deve devolver um objeto contendo as previsões dos recursos para a nova submissão.

ResourceMonitor Localizado no provedor de recursos, trata da monitoração de CPU e memória na máquina local.

A previsão ocorre no momento em que o gerenciador do aglomerado procura os nós candidatos a executar o trabalho, e somente se o usuário não fornecer uma estimativa própria. Nesse momento, a classe de previsão correspondente ao método escolhido devolve um objeto contendo um tempo de execução normalizado e um valor de memória. Essa é uma previsão genérica, que ainda precisa ser modificada de acordo com o nó que receberá o trabalho, devido à heterogeneidade da grade, e de acordo com o formato de entrada do LUPA. Vale notar que é preciso haver um número mínimo de casos passados, pois não faz muito sentido basear-se em uma série muito curta para gerar uma previsão. Por esse motivo só realizamos previsões quando há, no mínimo, 15 execuções similares armazenadas no banco de dados.

As modificações são feitas ainda no gerenciador do aglomerado, que modifica o tempo normalizado pelos BogoMips de cada nó candidato, obtendo uma previsão de tempo de execução em horas. A partir desse ponto o escalonamento é feito da mesma maneira, excluindo os nós que não garantiram os níveis de processamento e memória naquela janela de tempo.

Ao término do trabalho o InteGrade agrupa as informações da execução e se prepara para enviá-las à base de dados. Isso só acontece, contudo, se a execução terminou com sucesso e em mais de 5 minutos. Esse limite foi escolhido pois também é a granularidade em que o LUPA trabalha com suas previsões e execuções muito curtas não estão no escopo da previsão.

6. Conclusão

Apresentamos nesse trabalho um modelo simples dedicado a prever os recursos utilizados por aplicações em uma grade oportunista e sua implementação no InteGrade. Sua importância reside em retirar dos donos de aplicações a responsabilidade de estimar os recursos que serão utilizados por suas aplicações, fazendo com que o potencial da previsão de padrões de uso da máquina, tema de vários estudos anteriores e em andamento, seja plenamente explorado.

Dois métodos foram implementados, o mais conservador, que utiliza intervalo de confiança, pode ser particularmente interessante por apresentar menor probabilidade de ocorrência de cenários em que o usuário local sente degradação no desempenho do seu computador. Por outro lado, o outro método, que utiliza mediana, apresentou menor erro médio.

Simulações utilizando dados de *clusters* em atividade apontaram erros comparáveis ao de outros trabalhos da literatura. Comparando com o modelo mais preciso, nosso erro é no máximo 62%, com a mediana, e 128%, com intervalo de confiança, pior. Em outros casos, porém, apresentamos melhores resultados, chegando a um erro médio 71% menor. Esse é um bom resultado, especialmente porque a pretensão inicial não era obter um modelo mais preciso, mas sim criar um sistema simples, porém funcional e integrado ao resto do projeto.

Além dos resultados teóricos, a implementação representa uma grande parte desse trabalho. Um dos resultados desse trabalho é um novo *branch*, baseado na atual versão do InteGrade, com todo o sistema de previsão implementado e funcional, pronto para ser integrado *trunk* e lançado na próxima versão.

Infelizmente não foi possível realizar experimentos mais amplos utilizando essa nova versão. Isso se deve à falta de tempo e à complexidade da tarefa, pois é preciso, entre outros pequenos desafios, preparar uma grade oportunista, realizar o treinamento dos padrões de uso utilizando o LUPA, definir os algoritmos de escalonamento e encontrar um conjunto representativo de aplicações a serem executadas, possivelmente com comportamentos bem distintos de uso de processamento e memória. Além disso, a análise dos resultados deve ser feita com cuidado, pois a eficácia dos experimentos não depende apenas da previsão, mas também do LUPA, do escalonador e do trabalho em conjunto desses três sistemas.

Referências

Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload`.

Repositório integrade. `bzr://www.integrade.org.br/integrade/branches/resourcePredictor`.

Tao trading service documentation. `http://www.dre.vanderbilt.edu/~schmidt/DOC_ROOT/TAO/docs/releasenotes/trader.html#Constraints`.

(2010). Integrade. `http://www.integrade.org.br`.

Bezerra, G. C. (2006). Análise de Conglomerados Aplicada ao Reconhecimento de Padrões de Uso de Recursos Computacionais. Master's thesis, Departamento de Ciência da Computação - Universidade de São Paulo. Orientador: Marcelo Finger.

Conde, D. (2008). Análise de Padrões de Uso em Grades Computacionais. Master's thesis, Departamento de Ciência da Computação - Universidade de São Paulo. Orientador: Marcelo Finger.

de Mello, R. and Yang, L. (2009). Prediction of dynamical, nonlinear, and unstable process behavior. *The Journal of Supercomputing*, 49(1):22–41.

Dodonov, E., de Mello, F., et al. (2007). A Model for Automatic On-Line Process Behavior Extraction, Classification and Prediction in Heterogeneous Distributed Systems. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 899–904. IEEE Computer Society.

Finger, M., Bezerra, G. C., and Conde, D. M. R. (Accepted, 2009). Resource use pattern analysis for predicting resource availability in opportunistic grids. *Concurrency and Computation: Practice and Experience*.

Goldchleger, A. (2004). Integrade: Um sistema de middleware para computação em grade oportunista. Master's thesis, Departamento de Ciência da Computação - Universidade de São Paulo.

Goldchleger, A., Kon, F., Goldman, A., Finger, M., and Bezerra, G. C. (2004). InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459.

Li, H. (2007). Machine learning for performance predictions on space-shared computing environments. *International Transactions on Systems Science and Applications, invited paper*.

Liu, C., Yang, L., Foster, I., and Angulo, D. (2002). Design and evaluation of a resource selection framework for grid applications. volume 0, page 63, Los Alamitos, CA, USA. IEEE Computer Society.

Nassif, L., Nogueira, J., Karmouch, A., Ahmed, M., and de Andrade, F. (2007). Job completion prediction using case-based reasoning for grid computing environments. *Concurrency and Computation: Practice and Experience*, 19(9).

Smith, W., Foster, I., and Taylor, V. (1998). Predicting application run times using historical information. *Lecture Notes in Computer Science*, 1459(122ff):183.

VIII Workshop em Clouds, Grids e Aplicações

♦

Sessão Técnica 3

Escalonamento e Recursos

Toward Advance Resource Reservation in Mobile Grid Configurations Based on User-Centric Authentication

Matheus A. Viera[1], Cristiano C. Rocha[1], Michael A. Bauer[2], Miriam Capretz[3], M. A. R. Dantas[1]

[1]Dept. de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 88040-900 – Florianopólis – SC – Brazil

[2]Dept. of Computer Science – University of Western Ontario, Canada

[3]Dept. of Electrical & Computer Engineering – University of Western Ontario, Canada

{matviera,crocha}@inf.ufsc.br, bauer@csd.uwo.ca, mcapretz@eng.uwo.ca, mario@inf.ufsc.br

Abstract. *There has been growing interest in the use of mobile devices for providing access to the various applications, services and resources within grid computing environments. As is the case for most access to grid resources, the use of mobile devices requires user authentication. Given the limited amount of power available for mobile devices, the applications on these devices, including the authentication services used to access grid resources, must be designed to conserve power. In this paper, we present an authentication architecture utilizing a "lightweight" user-centric authentication approach. Specifically, we aim to provide mobile users with access to advance resource reservation within grid environments, and, consequently, our approach incorporates this objective. The approach, however, applies to user authentication with any grid resource or service. In attempting to overcome the limitations of mobile devices, such as limited battery power, mobile users can utilize grid environments in a transparent and secure way.*

1. Introduction

A Grid computing is characterized by making a variety of distributed resources, including services, devices and applications, available to a wide range of users [Foster 2002]. Various organizations, both real and virtual, can make different types of resources available under dynamically changing availability constraints and with varying policies for access and use of these resources [Foster et al. 2001]. Subsequently, the resources belonging to these organizations can be accessed and combined by different users to achieve their computational goals.

As a result of their nature, grids provide a variety of services that users can incorporate for achieving their computational tasks. Specifically, users can access resources, applications and services, submit jobs for execution either via queues or by advance reservation, create combination processes in workflows, and verify the status of jobs or systems. However, access to most grid environments requires some form of security and user authentication. In this context, authentication involves the process of

establishing the digital identity of a user or object to use the network, applications or resources. Once authenticated, a user can access the resources based on the permissions that they have been granted, which are established through the authorization process.

In recent years, there has been a movement towards integrating grid computing environments with mobile computing environments [Rossetto et al. 2007; Chu and Humphrey 2004; Gomes et al. 2007]. Consequently, mobile devices within this context are considered as grid interfaces and as grid resources. Despite the fact that the computing power of mobile devices has improved significantly in recent years, the current processing power and storage capacity found in these devices are still not enough to solve complex problems. Therefore, the present study considers the use of mobile devices as interfaces to access the resources and services of a grid from anywhere, at anytime. Our approach aims to enable users to utilize mobile devices for access to advance reservation services with the objective of submitting individual tasks and workflows. As is the case with other access to grid services, the use of mobile devices also requires a user authentication mechanism, which is a device that allows users to adopt defined or permitted roles for access to the services and resources.

As studies in area of mobility suggest, a change in the approach of performing user authentication via mobile applications, namely, from a process-oriented paradigm to a user-centered one, must be accomplished [Saha and Mukherjee 2003]. Specifically, the authentication system should recognize the user rather than the equipment that the user possesses. Moreover, because mobile devices have limited power, authentication schemes that are computationally intensive or that require substantial communication are unsuitable. This change to a user-centered approach, coupled with the limited power resources of mobile devices, imposes new requirements on the security and authentication systems for supporting the use of mobile devices within grid environments.

The current work presents an architecture that provides a "lightweight" user-centric authentication mechanism for the use of mobile devices within grid environments. In particular, its purpose is to provide the user with the full range of mobile service offered by these environments. Accordingly, our approach to authentication is in the context of providing the mobile user with access to resource reservation services.

The paper is organized as a series of sections; the motivation for the development of our approach is presented in Section 2. The proposed architecture is introduced in Section 3, and Section 4 presents the experimental results. Finally, the paper presents our conclusions and future research work in Section 5.

2. Motivation

The research in [Rossetto et al. 2007], a previous work of our group, proposes a framework for submitting and monitoring grid computing tasks through mobile devices. In that study, there is a mechanism for managing disconnections that result from a drop in battery power or from interference in the wireless network. However, this framework poses a disadvantage in the case where a user accesses different devices during the execution of an application. In this situation, each device must perform the entire authentication process, thus reducing the battery charge and the system productivity.

Furthermore, this work does not examine access to an advance resource reservation facility in the grid environment and uses a traditional authentication where a username and password are requested in each interaction.

Advance resource reservation in the grid computing environment has been the focus of recent research (e.g., [Takefusa et al. 2008], [Siddiqui et al. 2005] and [Roblitz and Reinefeld 2005]). By reserving resources, a client has guaranteed access to a specific resource in the grid for a designated time period. Accordingly, [Takefusa et al. 2008] and [Siddiqui et al. 2005] present mechanisms for resource reservation using the concept of co-allocation [Foster et al. 1999], which can be defined as the simultaneous use of grid resources across multiclusters. The approach described in [Takefusa et al. 2008] provides resource reservation for a single resource or for a set of resources by using co-allocation. In addition, the work suggests the use of a ticket for subsequent interaction with the reservation. In [Siddiqui et al. 2005], the authors present an approach that uses the Web Service Resource Framework (WSRF) [Globus Alliance 2010] to perform resource reservation, along with the introduction of a two-phase commit protocol. Their objective is to use a non-blocking mechanism that avoids disconnection problems and can facilitate the recovery of failed processes. While [Takefusa et al. 2008] suggests the use of co-allocation, [Roblitz and Reinefeld 2005] introduces the idea of virtual resources without co-allocation. The elimination of this mechanism allows the generic integration of different types of resources and reservations with the use of temporal and spatial relationships between components.

On the other hand, because of the movement from a process-based paradigm to the user-centric paradigm, some studies present requirements that must be considered for user-centric security systems. Therefore, in order to fulfill these requirements, it is necessary to analyze context-related information, such as the user's location, the time in which the interactions occur and the context-based detection of anomalous transactions [Mashima and Ahamad 2008]. According to Johnson [Johnson 2009], in highly dynamic environments as mobile grid environments, context-aware security mechanisms are necessary because the change of context is used by these mechanisms to allow the adjustment based on the current situation. Therefore, these mechanisms are able to effectively circumvent limitations of traditional security systems that are designed for static environments and are not suitable for the mobile computing paradigm. Several works present complex solutions for user authentication based on the environmental context [Choi and Yi 2008; Babu and Venkataram 2009]. Most of these studies achieve their objectives by using several sensors in these environments. However, these proposals restrict user mobility because they are only effective within the area covered by the sensor network. For instance, [Choi and Yi 2008], presents an architecture that aims to authenticate users based on the context captured by various sensors and devices present in the "smart homes" environments. Additionally, [Babu and Venkataram 2009] proposes an authentication scheme regarding the analysis of the user behavior. Even tough this study presents a module concerned with the power consumption of mobile devices regarding the required level of authentication, it is not able to consider the device switching that can be performed in mobile environments.

Nevertheless, these studies neglect the possibility of advance resource reservation and the possibility of monitoring an application through mobile devices. In addition, there are several research efforts to improve mobile environments, but none of

these studies present user-centric authentication methods or consider alternatives for reducing battery consumption. Since frequent problems, such as the disconnection of mobile devices in wireless networks could lead to application faults and high energy consumption, it would be necessary to run the authentication process over again, leading to more power usage.

3. Proposed Architecture

In this section, we propose a user-centered architecture that enables advance resource reservation in grid environments. Our approach is partially based on the research presented in [Rossetto et al. 2007], where the authors propose a framework for the submission, monitoring and coordination of workflows executed in grid computing environments. All of the interaction with the grid is performed through mobile devices that are used as grid interfaces. In particular, this work suggests the possibility of adapting the execution flow to guarantee the consistency of an application in case of a disconnection occurs. This execution flow will be performed in a personalized way in the case that the mobile device is disconnected from the network regarding the dependences among the tasks. Specifically, these features are performed by **Workflow Manager, Agent** and **Mobile GUI.**

The **Workflow Manager** module is responsible to manage the requests processes from mobile devices in a transparent way to the users. In particular, it provides an automated way of submitting jobs to the grid and it collects information about the execution of these jobs without user interaction. Thus, this mechanism contributes to the reduction of battery power consumption in mobile devices. Also, the architecture provides a mechanism for fault tolerance, especially in the case where a voluntary or involuntary disconnection of mobile devices occurs. **The Agent** module is responsible for verifying when the disconnection occurs; if the device is connected, it uses a specific time interval. Additionally, this module also manages the faults by detecting the failure and adapting the application execution flow to the environment. When a disconnection occurs, **The Agent** can adapt to the situation by continuing the execution, waiting for device establish a connection or aborting. These options are defined by the user, and the actions are performed according to the existence of dependencies from the user. Hence, through this module, the consistency of the application is guaranteed.

All grid computing interactions occur through a **Mobile GUI** interface. This interface is responsible for allowing workflow submission and permitting the visualization of final and partial results of the application in an optimized way, since only the parts of the resulting files that are considered relevant for the user are loaded in the device interface. Also, the interface contains the ability for users to monitor the application execution, so that the status of each task can be traced. In addition, users can monitor the application execution in a customized manner based on the type of mobile device and its particular screen size. Finally, the **Mobile GUI** is responsible for sending the username and password to **Workflow Manager** for authenticating the user on the grid environment. Consequently, the user has to authenticate every interaction with the grid through their mobile device.

In addition to **Workflow Manager, Agent** and **Mobile GUI**, as proposed by

Rossetto et al. [Rossetto et al. 2007], the modules of **User-centric Authentication** and **Resource Reservation** were added in the present approach for enabling a more secure and more efficient interaction in mobile grid environments. The new modules attempt to take advantage of the mobility offered by mobile devices while utilizing the resources from grid environments more safely and effectively. In addition to these goals, our approach attempts to improve the battery consumption of mobile devices.

Figure 1 illustrates the research architecture of [Rossetto et al. 2007] with the addition of the new modules, User-centric Authentication and Resource Reservation, which are added to the proposed framework.

3.1. User-Centric Authentication

The mobile computing paradigm suggests that the procedures are focused on the user regardless of the device that he/she is using. Therefore, the objective of the user-centric authentication module consists of providing the user with the advantages offered by mobile devices, such as Personal Digital Assistants (PDAs) and smartphones, in grid environments. These advantages include the possibility of anytime and anywhere access to the grid. Moreover, grid access in a safe, transparent and automatic manner is also part of the current work.

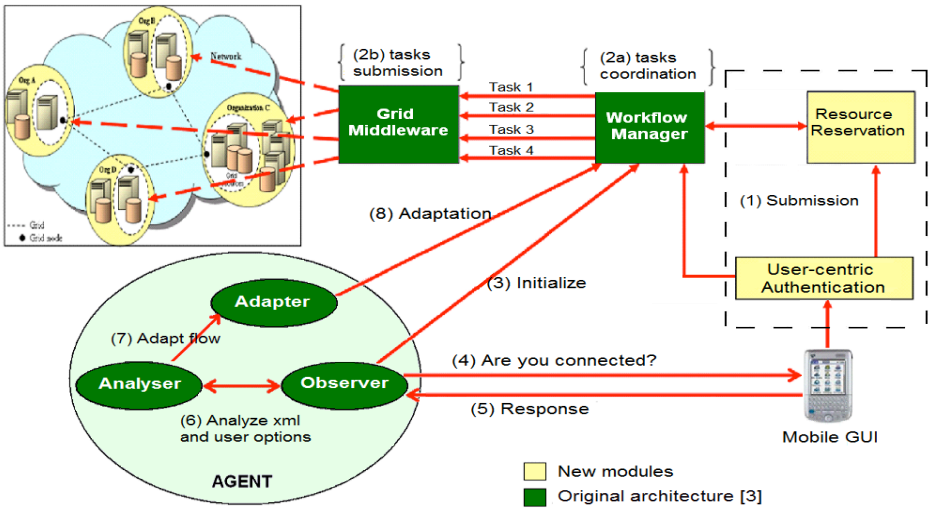


Figure 1. Proposed architecture

Through device independence, which is referred to as user-centric computing throughout this paper, the user can switch from an equipment to another without requiring a new authentication process. Thus, the potential problem of insufficient battery power can be avoided by device replacement. Moreover, the user's interaction with an application communicating with the grid environment is not interrupted, nor is it necessary to restart the authentication process. The authentication module is responsible for the interception of all service requests, which are workflow submissions or advance resource reservations requested by an individual using a mobile device. This authentication approach obtains transparency by using a widely disseminated standard among mobile devices: the vCard standard [Dawson and Howes 1998], which aims at the automation of personal information sharing frequently found in a regular identification card. In this standard, the data is presented by using pre-defined meta-

information that is responsible for receiving data in an organized manner and then facilitating the utilization of the data. The standard has been maintained by the Internet Mail Consortium (IMC) since 1996; the vCard standard is compatible with several existing platforms and is mainly concentrated on mobile devices, such as cell phones and PDAs.

Furthermore, the vCard standard permits the extension of meta-information in order to store other necessary data. Thus, it permits greater flexibility in the manipulation of user information and in the adequacy of each required application. The inserted information does not affect the original standard because it is ignored by the interpreter. Subsequently, the unity guarantee is maintained among various applications that involve the exchange of the same electronic identification card. Furthermore, this standard provides security for the stored information, because it offers support for digital signatures.

Therefore, the authentication system uses the vCard standard to store system-specific information. This information, such as environmental access credentials, or tickets, is represented by strings that are created by the system and that have a predetermined duration, which depends on the permissions granted to the user. For a user that has permission to reserve resources, the ticket could expire at the time of the reservation. Thus, while the lifetime of the user ticket is limited, the user can utilize other devices that have the properly formatted vCard in the system. In other words, the device has the vCard extended, so that the user does not need to reinsert his access data in the system. Figure 2 shows an example of credential represented by the vCard standard.

```
BEGIN:VCARD
VERSION:2.1
FN:Cristiano Rocha
N:Rocha;Cristiano;;;
ORG:LaPeSD
TICKET:2e8475e3c149f73e8f56bca51377a7e2
ADR;TYPE=work:;;;Florianopolis;SC;Brasil;
EMAIL;TYPE=internet,pref:crocha@inf.ufsc.br
REV:20090616T150922Z
END:VCARD
```

Figure 2. Example of an extended electronic identification card represented in the vCard format

The user-centric authentication module and its interactions with the other system modules are shown in Figure 3. The authentication process is used to prove the digital identity of the user. When the user requests a service, the system accesses the vCard in the device. (I) The **Credential Manager** selects the specific credentials of the application that are contained in the vCard and verifies if the user is on the list of active users; in other words, it verifies if the ticket is still valid. If the ticket has expired, (II) the **Credential Manager** queries the **Device Manager** to verify if the user switched to a different device. If a switch has occurred, (III) the **Location Manager** performs the functions illustrated in the activity diagram in Figure 4. Otherwise, it determines if there is an association between the user and the device used in the request. If such an association exists, the **Ticket Manager**, which controls the lifecycle of the tickets, creates a new ticket. Consequently the **Credential Manager** updates the vCard with the new ticket, and subsequently, the updated vCard is stored in the **Credential**

Repository, and it is also sent to the device. Otherwise, if the user cannot be associated with the device, the system requests the login and the password of the user. (IV) If the ticket is still valid, the **Permission Manager** is queried in order to verify that the user has the permission for the requested operation. If the user has the appropriate permission, (V) the request is sent to execute the requested operation. This operation is one of the available services, along with resource reservation, reservation cancelation and the submission and monitoring of downloaded application results, the latter of which is also provided in [Rossetto et al. 2007]. Finally, the operation response requested by the user is returned to him/her (VI).

Moreover, since one of the main goals of the user-centric authentication module is the securityof the user's information in the environment, all messages sent between themobile devices, **Mobile GUI**, and the **Credential Manager** are encrypted. Therefore,this procedure tries to prevent malicious users from acquiring unauthorized access togrid services.

In recent years, mobile devices are able to perfom geometric models used to determine object location with geographic coordinates. These models are commonly used by location models based on the GPS (Global Positioning System). Thus, due to the advantages offered by applications that are able to facilitate location-related queries and manage accurate information regarding the location of objects, the latest generation mobile devices is being equipped with GPS to provide support for these applications. Also, it is possible to configure the accuracy of the location in order to safe battery of mobile devices. Therefore, the integration of these two popular technologies, vCard and GPS, which is still under development in the authentication system, is responsible to improve the security offered to users in the environmnent .The next section describes the functionalities of the authentication system regarding the user's location.

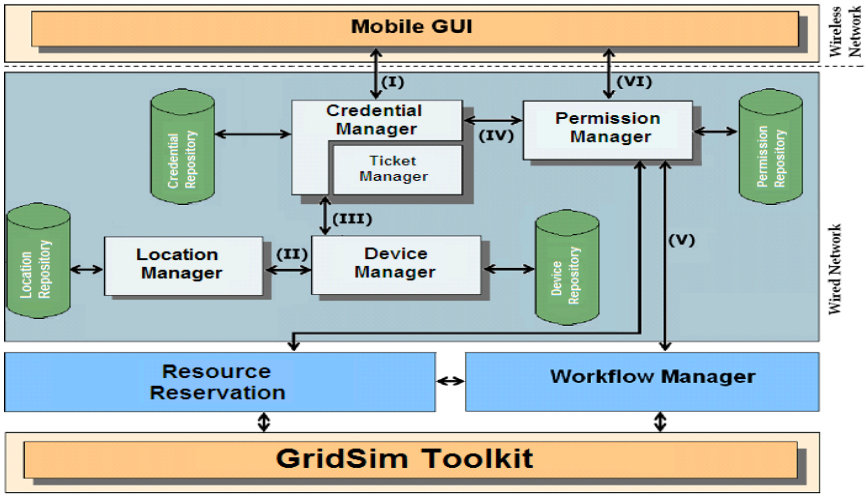


Figure 3. Authentication module architecture and its interaction with the others modules

3.1.1. Spatio-Temporal Analytical Model

In order to provide more reliability to the mobile grid environment as well as to minimize and detect fraudulent activities, the authentication system considers the capacity of the mobile devices to capture information regarding the spatio-temporal context of the environment where they are inserted. Thus, the **Location Manager** can

classify the performed activity (event) regarding the geographic location and the time frame of occurrence of this event simultaneously. In order to formally define an event, we assume that an event is an interaction (activity) of the user with the application or environment in a certain location and at a particular time frame. Then, an event is described as:

$$E_i =< operation, time, location >$$

Therefore, the observed events in the execution of activities form a database (**Location Repository**) for the process of detecting information clusters, which translates to the behavior of users. These clusters can be classified into three broad categories: purely spatial, purely temporal or spatio-temporal. In purely spatial clusters, the occurrence is higher in some regions than it is in others, and purely temporal clusters feature the occurrence of events as being greater in a certain period than it is in others. Finally, spatio-temporal clusters occur when events are temporarily higher in certain regions than they are others. Among the models used to predict events in a spatio-temporal context, we propose the use of spatio-temporal permutation, which allows the incorporation of covariate information found in other contexts within the pervasive space. The Poisson model, which is applied to purely temporal contexts, and the Bernoulli model, which is preferably applied to spatial contexts, were both rejected because they do not consider both the location of the user and the time frame during an occurrence of an event.

According to Kulldorff [Kulldorff 2010], the spatio-temporal permutation model is based on three characteristics: *i)* detecting data clusters in space and time simultaneously; *ii)* working with only events or cases; and *iii)* applying the probabilistic model in a null hypothesis to conclude that the events follow a hypergeometric distribution.

Assuming the count of events $\boldsymbol{e}$, in the timeline set in $\boldsymbol{t}$, located in a region $\boldsymbol{z}$, with circular features according GPS coordinates, is defined as $\boldsymbol{e_{zt}}$. The total number of observed events $\boldsymbol{E}$ and the total number of conditioned events $\boldsymbol{M_{zt}}$ are expressed by the following formulas:

$$E = \sum_z \sum_t e_{zt} \qquad\qquad M_{zt} = \frac{1}{E}\left(\sum_z E_{zt}\right)\left(\sum_t E_{zt}\right)$$

The prediction of an event encompasses the following assumption: the conditional probability of an event $\boldsymbol{P(E_a)}$ in the region $\boldsymbol{z}$ was observed at the time $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$, defined in a particular cylinder $\boldsymbol{a}$, which reflects a possible cluster; therefore $\boldsymbol{E_a}$ has an average $\boldsymbol{M_a}$ and follows the hypergeometric distribution determined by the following function:

$$M_a = \sum_{(z,t)\in A} M_{zt} \qquad\qquad P\left(E_a\right) = \frac{\left(\begin{array}{c}\sum\limits_{t\in(t_1 \vee t_2)}\sum\limits_{z\in A} E_{zt} \\ E_a\end{array}\right)\left(\begin{array}{c}E - \sum\limits_{t\in(t_1 \vee t_2)}\sum\limits_{z\in A} E_{zt} \\ \sum\limits_{t\in(t_1 \vee t_2)}\sum\limits_{z\in A} E_{zt} - E_a\end{array}\right)}{\left(\begin{array}{c}E \\ \sum\limits_{t\in(t_1 \vee t_2)}\sum\limits_{z\in A} E_{zt}\end{array}\right)}$$

In order to determine the regions of the clusters, it will be used the *SaTScan* tool developed by Kulldorff [Kulldorff 2010] and the statistical significance will be

validated by using the hypothesis test of Monte Carlo.

The conditional probability of the user $P(E_a)$ allows the system to estimate what kind of activity the user was performing and what one he is currently performing when he moves from a mobile device to another one. Thus, there are four cases that can occur: *i)* the same activity in the same spatio-temporal context – it is defined as a normal execution; *ii)* same activity in different spatio-temporal contexts – it is defined as a suspicious execution, but some properties must be considered such as the velocity of mobility in order to apply the appropriate authentication policies; *iii)* different activities in the same spatio-temporal context – it is defined as a suspicious execution; and *iv)* different activities in different spatio-temporal context – it is defined as an abnormal execution. Therefore, depending on the categorization of the user, the authentication system defines which action will be taken regarding the following factors: the performed request, the malicious user, the mobile device and the potential fraud victim. The activity diagram shown in Figure 4 illustrates how the system operates when it detects device switching.

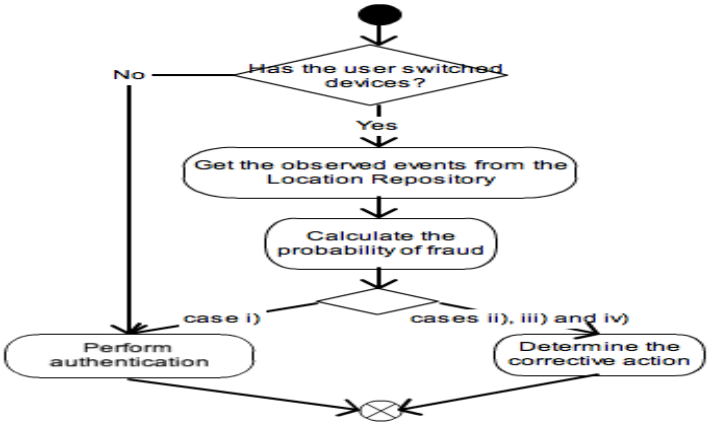


Figure 4. Fraud detection activity diagram

3.2. Resource Reservation

As previously discussed, the possibility of resource reservation allows mobile users to plan their future use of the grid. The resource reservation module, which is still under development, is responsible for ensuring that these reservations are maintained for future workflow submission on the grid. In addition, it enables the monitoring of reserved resources as well as recording any cancelled reservations. Figure 5 illustrates the design of the module and its functionality is described in the subsequent paragraphs. First, after intercepting the request and authenticating the user, the authentication service transfers the ticket access information and the resources requiring reservation to (I) the **Reservation Service**. At the same time, the authentication module transfers the start time and end time of the reservation to the **Reservation Service**, which uses (II) the **Grid Information Service (GIS)** to verify the availability and status of the requested resources. If there are available resources, (III) the **Co-allocation Service** will select the best option based on the information from the **GIS**, and it will allocate the resources accordingly. At this point, the user ticket is associated with the reservation, which enables future interactions between the user and the system, allowing the user to verify the status of reservations, cancel a reservation or monitor the workflows. Subsequently, information pertaining to the allocated resources and the user ticket are

stored in (IV) a **Data Base** (DB), hence enabling a checkpoint mechanism. This method is necessary in case a user wishes to access a previous workflow result or interact with the grid environment to submit or cancel workflows.

The ticket created by the authentication module might specify a duration time based on the time of the reservation. When a ticket is no longer valid, the resources reserved by the user are automatically released. When the user submits a workflow to the environment, the Reservation Service searches the information in the Database to verify the user ticket. Once the ticket is verified, the workflow is sent to the **Workflow Manager** (V).

Through the reservation module, users can plan the future use of resources based on their mobile requirements. Since the reservation is performed and monitored through their mobile device, the user does not need to worry about which device will make submissions and monitor workflows. Rather, they need to ensure that the resources have been previously allocated.

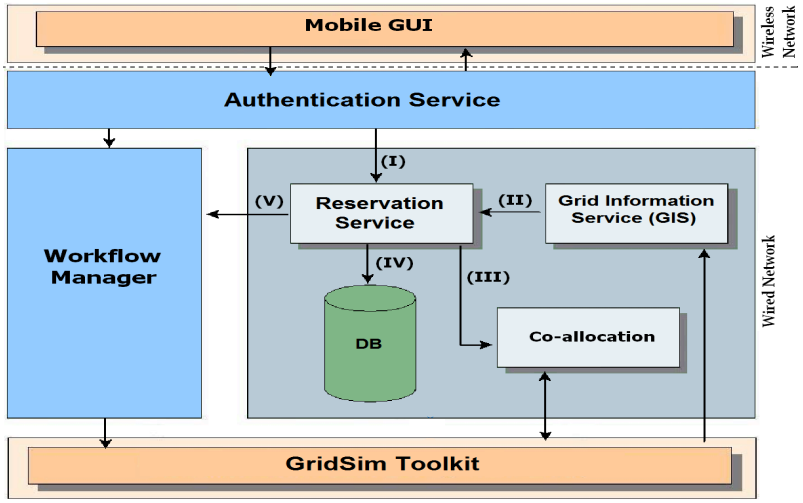


Figure 5. Resource reservation architecture

The development of the resource reservation module was enabled by GridSim [Sulitio and Buyya 2004]. The toolkit supports modeling and simulation of heterogeneous grid resources, users and application models. This toolkit will assist with the implementing the Reservation Service component, modeling heterogeneous computational resources of variable performance, and testing the policy of advance resource reservation.

4. Experimental Results

Our experiments were performed on the basis of the environmental configuration proposed in [Rossetto et al. 2007] and focus on the battery usage without concern about potential security threats. Therefore, the Java programming language was used for implementing the user-centric authentication service and integrating it with the other modules. In addition, the module present in the Mobile GUI, which is responsible for handling the user's vCard and intercepting service requests, was implemented using the J2ME (Java 2 Micro Edition) Wireless Toolkit. In addition, the simulator GridSim was used as the grid environment.

The experimental environment consisted of a server containing the

authentication service, which was integrated with the other modules presented in [Rossetto et al. 2007]. Also, the mobile devices used in our experiments were two Palm Tungsten C devices, each with a 400MHz processor, 64 MB RAM, built-in Wi-Fi (802.11b), and a Palm OS 5.2.1 operating system. As the integration of GPS with authentication system is still under development, for the current experiments we used a device without GPS.

Since the maintenance of battery life is one of the most critical and challenging problems found in mobile devices, such as PDAs and cell phones [Mohapatra et al. 2005; Rong and Pedram 2003], new methods and techniques are required to reduce the dissipation of energy in such devices. Accordingly, we analyzed the efficiency of the proposed user-centric authentication mechanism based on the power consumption of mobile devices.

This analysis was performed by identifying a pre-defined sequence of ten service requests. Specifically, this analysis compared the execution of a sequence using the user-centric authentication approach and the traditional authentication approach. The traditional authentication approach refers to the device-centric authentication method used in [Rossetto et al. 2007], where a username and password are requested when the user moves from one mobile device to another. The device-centric authentication was chosen for analysis because it is one the most common authentication mechanisms in mobile grid environments, as indicated in the research. In order to evaluate the efficiency of the mechanism proposed in this paper, we simulated a user changing mobile devices. Therefore, the pre-defined requests were interspersed between the two devices by performing the pre-defined sequence in a mobile device, then running it in the other one.

Figure 6 presents empirical results using the battery consumption of the mobile devices as the metric for the performance of the proposed approach, which compared user-centric authentication to the traditional authentication. This evaluation was performed using the BatteryGraph software [Witteman 2010], which had been installed on the mobile devices. It provides the mean battery level, expressed in milivolts (mV), before and after the completion of each of the two applications.

Figure 6 indicates that the user-centric authentication approach demonstrates a consistent increase in power consumption based on the number of requests. In comparison, the traditional authentication approach causes a greater increase in the energy consumption of the battery. Thus, the proposed approach represents a significant reduction in the power consumption of the battery as compared to the traditional authentication mechanism.

The second experiment attempted to analyze the efficiency percentage of the proposed mechanism. Specifically, it consisted of running the same sequence of pre-defined requests several times until the battery was totally depleted. First, this procedure was performed using the traditional authentication approach and switching the mobile devices between the sets of requests. Subsequently, the same experiment was performed by executing the application with the user-centric authentication mechanism. As in the case of the traditional approach, this execution also used the process of interspersing the sets of requests between the two mobile devices. In addition, in order to acquire an accurate assessment, both experiments were performed three times.

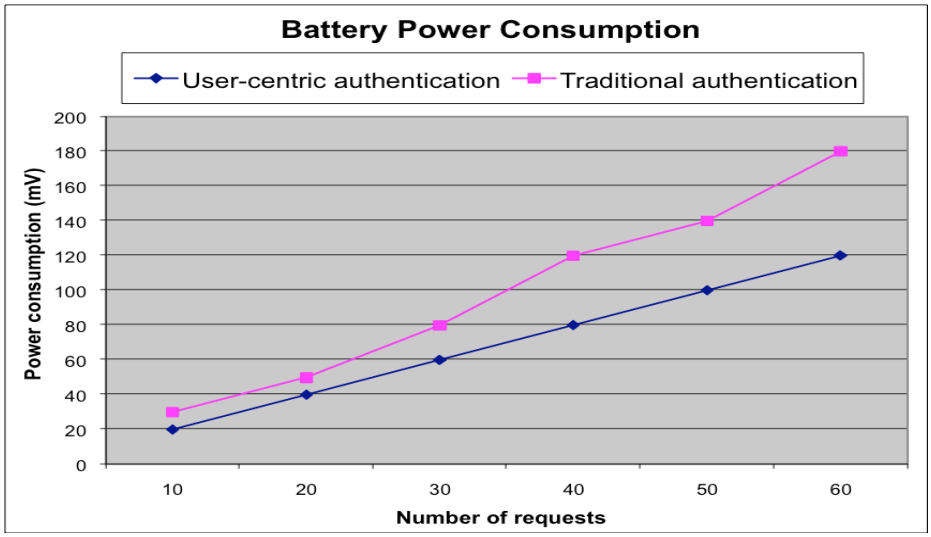


Figure 6. Battery power consumption for two authentication approaches

Figure 7 presents the results obtained by using the two authentication approaches. The BatteryGraph was utilized for obtaining the percentage of the devices' charge level. Moreover, it also verifies the percentage of battery used during a time interval selected by the user.

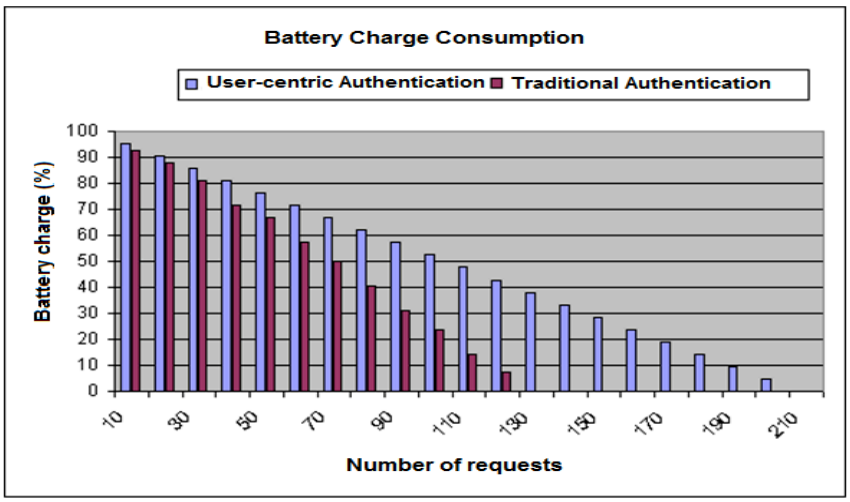


Figure 7. Comparison of battery charge for two authentication approaches

As shown in Figure 7, the mechanism proposed in this work for user-centric authentication enables a greater autonomy of energy in comparison to the traditional authentication mechanism. For instance, the traditional authentication mechanism resulted in total battery exhaustion after 130 requests, while the proposed approach for user-centric authentication did not completely drain the battery until after 200 requests. Therefore, the user-centric authentication approach results in a noticeable increase of approximately 53% in the battery life.

5. Conclusions and Future Work

This paper proposed an approach for enabling safe advance resource reservation in mobile grid environments by adopting the user-centric authentication approach. The proposal addressed the shortcomings in [Rossetto et al. 2007], which were mainly due

to the inefficiency of various components in the environment. This work aimed to create a safe and transparent system for users to submit tasks and reserve resources in mobile grid environments. Specifically, its primary objectives consisted of making the user's interaction with the environment more flexible and reducing the battery consumption of mobile devices, both of which were successfully achieved. In addition, the proposal also aimed to provide to the user with more efficient mobility resources in such environments.

Although it is recognized that there is a need for experiments focused on other relevant requirements in mobile grid environments such as interactivity of the user, behaviour of the authentication service when there are disconnections and interferences in the wireless network and the ability of the system in detecting frauds. This study has demonstrated that the proposed approach for user-centric authentication illustrates the improved performance of battery life in comparison to the traditional authentication approach for mobile grid environments. This improvement indicates that a significantly greater number of operation requests can be performed by the user that operates multiple mobile devices during his interaction with the environment. Therefore, the approach proposed in this paper provides a more productive environment for the user.

As an extension of the current research, we intend to perform further simulations using other mobile device models in order to perform experiments regarding the user's location and the impact on the experience of the user. Moreover, we will consider other important metrics in mobile grid environments, such as the occurrence of disconnections and interferences in the wireless network, user interactivity, and the ability of the system in detecting frauds. In addition, we intend to evaluate the latency of the user-centric authentication approach, particularly, the way in which it affects the resource reservation environments. Although this implementation only represents one type of improvement, we believe that the design implications presented in this study are applicable to other scenarios.

References

Babu, S. and Venkataram, P. (2009) "A dynamic authentication scheme for mobile transactions", International Journal of Network Security, vol. 8, pp. 59-74.

Choi, H. and Yi, Y. (2008) "An user-centric privacy authorization model based on role and session in the context-aware home", Proceedings of the 8th IEEE International Conference on Computer and Information Technology Workshops, pp. 254–259.

Chu, D. and Humphrey, M. (2004) "Mobile OGSI .NET: Grid computing on mobile devices", Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, pp. 182–191.

Dawson F. and Howes, T. (1998) "RFC2426: vCard MIME directory profile", RFC Editor, USA.

Foster, I. (2002) "What is the grid? A three point checklist", GridToday, vol. 1, no. 6.

Foster, I., Kesselman, C. and Tuecke, S. (2001) "The anatomy of the grid: Enabling scalable virtual organizations", International Journal of High Performance Computing Applications, vol. 15, no. 3, pp. 200-222.

Foster, I., Kesselman, C., Lee, C., Lindell, B., Nahrstedt, K. and Roy, A. (1999) "A distributed resource management architecture that supports advance reservations and co-allocation", 7th International Workshop on Quality of Service, pp. 27–36.

Globus Alliance (2010) "The WS-Resource Framework", http://www.globus.org/wsrf/

Gomes, A., Ziviani, A., Lima, L. and Endler, M. (2007) "DICHOTOMY: A resource discovery and scheduling protocol for multihop ad hoc mobile grids", Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid, pp. 719–724.

Johnson, G. (2009) "Towards shrink-wrapped security: A taxonomy of security-relevant context", Proceedings of the 7th IEEE International Conference on Pervasive Computing and Communications, pp. 1-2.

Kulldorff, M. (2010) "SaTScan v7.0: Software for the spatial and space-time scan statistics", http://www.satscan.org/

Mashima, D. and Ahamad, M. (2008) "Towards an user-centric identity-usage monitoring system", Proceedings of the 3rd International Conference on Internet Monitoring and Protection, pp. 47–52.

Mohapatra, S., Cornea, R., Oh, H., Lee, K., Kim, M., Dutt, N., Gupta, R., Nicolau, A., Shukla, S. and Venkatasubramanian, N. (2005) "A cross-layer approach for power-performance optimization in distributed mobile systems", Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, vol. 11, pp. 8.

Roblitz, T. and Reinefeld, A. (2005) "Co-reservation with the concept of virtual resources", 5th IEEE International Symposium on Cluster Computing and the Grid, vol. 1, pp. 398–406.

Rong, P. and Pedram, M. (2003) "Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach", Proceedings of the 40th Conference on Design Automation, pp. 906–911. ACM New York, NY, USA.

Rossetto, A., Borges, V., Silva, A. and M. Dantas (2007) "SuMMIT – A framework for coordinating applications execution in mobile grid environments", Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, pp. 129–136.

Saha, D. and Mukherjee, A. (2003) "Pervasive computing: a paradigm for the 21st century", IEEE Computer, vol. 36 no. 3, pp. 25–31.

Siddiqui, M., Villazon, A., Prodan, R. and Fahringer, T. (2005) "Advanced reservation and co-allocation of grid resources: A step towards an invisible grid", Proceedings of 9th IEEE International Multitopic Conference, pp. 1–6.

Sulistio, A. and Buyya, R. (2004) "A grid simulation infrastructure supporting advance reservation", Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems, pp. 1-7, MIT, Cambridge, USA.

Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y. and Sekiguchi, S. (2008) "GridARS: An advance reservation-based grid co-allocation framework for distributed computing and network resources", LNCS, 4942, pp. 152-168.

Witteman, J. (2010) "BatteryGraph", http://palm.jeroenwitteman.com

Um Estudo sobre Utilização do Consenso para Escalonamento de Tarefas em Grades Computacionais

José Souza de Jesus[1], **Genaro Costa**[1], **Fabíola Gonçalves Pereira Greve**[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
Salvador – BA – Brazil

{joss,genaro,fabiola}@dcc.ufba.br

Resumo. *Este trabalho tem como objetivo analisar a viabilidade e eficiência de uma abordagem baseada em consenso para escalonamento de alto nível de tarefas em grades computacionais, num cenário sem falhas. Tal análise considera quatro sugestões de algoritmo de escalonamento distintas na maneira de atualizar as informações utilizadas no escalonamento. Foram realizados testes envolvendo os algoritmos sugeridos, com base em informações extraídas de uma grade real para determinar as cargas de trabalho e os recursos da mesma. Os resultados obtidos, a partir de comparações entre o comportamento dos algoritmos propostos, demonstram que a abordagem baseada em consenso não traz benefícios num cenário sem falhas e tende a gerar latência, sobretudo quando o consenso envolve muitos participantes.*

Abstract. *This paper aims at evaluating the viability and efficiency of a consensus-based high-level job scheduling in computational grids in a fault-free environment. This study considers four proposals of scheduling algorithms, which distinguish themselves on how to update informations used for scheduling. Simulation tests were conducted involving the four proposals and based on informations obtained from a real grid. The results show that the consensus-based algorithm has no benefits in a scenario without failures and tends to generate latency, specially when consensus involves many participants.*

1. Introdução

Nos últimos anos, a computação em grades mostrou-se uma alternativa barata para a execução de aplicações que requerem alta capacidade de processamento e armazenamento. Uma grade computacional (ou simplesmente *grid*) é um sistema distribuído, com infra-estrutura de *software* e *hardware*, que proporciona acesso a capacidades computacionais de alto desempenho [Foster and Kesselman 1998]. Grades são formadas por recursos (máquinas) dispostos em domínios distintos, com suas próprias políticas de acesso. Esses recursos podem realizar computação local ou podem estar indisponíveis por certo tempo (não são máquinas dedicadas). Uma grade pode ser formada por milhares de máquinas distintas em arquitetura, sistema operacional, localização geográfica, dentre outras particularidades.

Para que seja possível a execução aplicações em grades, um mecanismo para gerenciamento de recursos e de tarefas deve ser provido de forma a garantir propriedades como disponibilidade dos recursos, segurança de dados e escalonamento das tarefas. Este

gerenciamento torna-se mais complexo com o aumento do número de recursos disponíveis na grade, influenciando na comunicação e transmissão de dados [Krauter et al. 2002].

Um aspecto importante no estudo de grades é o escalonamento das tarefas. Em primeiro lugar, o recurso para qual será alocada a tarefa deve ser capaz de executá-la. Além disso, informações sobre o estado do recurso, tais como disponibilidade, utilização e poder de processamento, podem ser de grande utilidade na realização de um escalonamento mais eficiente. Entretanto, obter e atualizar tais informações pode ter custo alto (uma vez que geram tráfego na rede e podem tornar-se obsoletas rapidamente). Essas informações podem ser fornecidas por um serviço do sistema, chamado *Grid Information Service* (GIS) [Czajkowski et al. 2001]. Em sistemas de larga escala, como o EGEE [Laure and Jones 2008], o tempo de propagação sobre o estado de ocupação dos recursos locais geralmente ultrapassa os dois minutos.

Outra questão pertinente no escalonamento diz respeito à tolerância a falhas dos recursos. Desta maneira, caso um recurso falhe (ou torne-se inacessível) deve ser garantido que as tarefas presentes no mesmo não sejam perdidas e/ou possam ser executadas por outro recurso. O consenso é um mecanismo fundamental para a construção de sistemas distribuídos confiáveis [Greve 2005]. Neste, um conjunto de processos deve escolher um valor dentre aqueles propostos pelos participantes. O consenso pode ser utilizado em problemas onde há a necessidade de construir uma visão idêntica do estado da computação para todos os processos ou para se ter uma ação coordenada destes processos, em determinado momento.

O uso do consenso em grades é defendido em [Hurfin et al. 2006], através do uso de um sistema de gestão de grupos que permite a organização dos recursos em domínios. O consenso é realizado de maneira a permitir um melhor escalonamento a partir da visão comum obtida do estado dos recursos e da sua disponibilidade para executar as tarefas requisitadas. Os autores argumentam que a falta de visão comum acarreta sobrecarga de recursos locais já que diferentes escalonadores selecionam o recurso mais livre para execução de suas tarefas, congestionando esse recurso. Uma visão consistente via consenso eliminaria esse tipo de problema.

Dando prosseguimento à proposta teórica lançada por [Hurfin et al. 2006], o objetivo deste trabalho é avaliar a viabilidade do uso do consenso para promover o escalonamento de tarefas numa grade computacional, levando-se em conta uma carga real da mesma. Ao nosso conhecimento, nenhum outro estudo experimental com este intuito foi efetuado. Para este estudo, propomos quatro algoritmos de escalonamento, distintos na maneira como as informações para escalonamento são atualizadas. Testes de desempenho para os algoritmos foram efetuados considerando-se um cenário sem falhas. Os resultados obtidos mostram que o uso do consenso nesse cenário não é aconselhável porque induz um aumento significativo na latência do escalonamento.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 define o modelo e o problema do consenso. A Seção 3 discute o escalonamento de tarefas em grades. A Seção 4 apresenta algumas propostas de algoritmos para escalonamento. Na Seção 5 são apresentados resultados de simulação e um estudo comparativo do desempenho das propostas. Por fim, a Seção 6 apresenta as conclusões do estudo realizado e delineia trabalhos futuros.

2. Problema do Consenso

Modelo do Sistema. Considera-se neste trabalho um sistema distribuído formado por um conjunto finito de n processos. Os processos são escalonadores que se comunicam através da troca de mensagens por canais confiáveis, onde não ocorrem perda ou corrupção de mensagens. O sistema é assíncrono, ou seja, nenhuma hipótese temporal existe no que diz respeito à realização das ações efetuadas pelos processos ou pelos canais.

Consenso. Informalmente, um grupo de processos atinge o consenso quando todos votam por um valor e os processos corretos (que não falharam) alcançam uma decisão comum sobre um dos valores propostos. Formalmente, para haver consenso, três propriedades devem ser garantidas [Chandra and Toueg 1996]:

- *Terminação*: Todo processo correto decide de maneira definitiva;
- *Acordo*: Dois processos (corretos ou não) não decidem diferentemente;
- *Validade*: Um processo decide no máximo uma vez e o valor decidido deve ser um dos valores propostos.

Em ambientes distribuídos assíncronos com falhas o consenso é impossível de ser obtido de maneira determinística[Fischer et al. 1985]. Isso decorre do fato de não ser possível diferenciar um processo lento de um processo falho. Algumas abordagens foram propostas para contornar tal resultado negativo, nas quais enriquece-se o sistema com algum grau de sincronia. Dentre essas abordagens, destaca-se os detectores de falhas não confiáveis [Chandra and Toueg 1996, Chandra et al. 1996]. Os detectores de falhas são oráculos distribuídos, acoplados a cada processo, que fornecem dicas sobre processos falhos. Eles são não-confiáveis porque podem cometer equívocos, suspeitando de processos corretos ou afirmando que processos faltosos são corretos. De certa forma, os detectores de falhas abstraem as características temporais necessárias para solucionar o problema do consenso num ambiente assíncrono.

Diversos problemas podem ser resolvidos (ou reduzidos) ao problema do consenso. Por exemplo, o serviço de gestão de grupos (ou *group membership*) utiliza-se de consenso para se manter uma visão uniforme da composição do grupo [Greve and Narzul 2004]. Numa grade, [Hurfin et al. 2006] propõem o uso do serviço de gestão de grupos tanto para controle do conjunto de domínios (controle inter-grade) quanto para controle do conjunto de recursos (controle intra-domínio).

Consideramos que o consenso pode ser usado em grades tanto para estabelecer uma visão consistente do estado de ocupação dos recursos computacionais, como para possibilitar uma estratégia global de escalonamento de tarefas. Nesse cenário, os escalonadores de alto nível fazem uso dessa visão consistente para evitar a sobrecarga de recursos, permitindo um melhor divisão da carga de trabalho entre os recursos.

3. Escalonamento em Grades Computacionais

Basicamente, escalonar uma tarefa é alocar a mesma para um recurso, após realizar um mapeamento entre as tarefas a serem executadas e os recursos disponíveis do sistema [Dong and Akl 2006]. Em grades, acontece de maneira análoga. O processo pode

ser definido em três estágios: (i) descoberta e filtragem dos recursos, (ii) escolha dos recursos segundo certos critérios (requisitos da tarefa, por exemplo) e (iii) submissão da tarefa [Schopf 2001].

Entretanto, o escalonamento de alto nível em grades deve lidar com algumas questões particulares. É importante notar que normalmente os recursos são gerenciados localmente em seus domínios, ou seja, os escalonadores não possuem controle total sobre os recursos, estando sujeitos a políticas de acesso distintas, assim como problemas decorrentes do acesso de um mesmo recurso por diversos escalonadores. Além disso, o comportamento da grade é dinâmico: tarefas em execução podem ser interrompidas por atividades locais; recursos ou domínios inteiros podem ingressar e sair da grade a qualquer momento; a carga sobre o recurso (tanto processamento quanto rede) pode variar bruscamente com o tempo; entre outras. Por fim, há a necessidade de se ter um processo rápido e transparente para o usuário.

Diversas soluções foram propostas para os problemas envolvendo escalonamento. Uma abordagem conhecida é baseada em fila de tarefas (*Work Queue*). Consiste em escolher as tarefas de maneira arbitrária e enviar para o recurso assim que estiver disponível, sem utilizar-se de qualquer informação sobre a natureza da tarefa. Neste caso, pode ocorrer de alguma tarefa mais robusta ser enviada para um recurso de baixa capacidade (resultando em longo tempo de processamento) ou até mesmo pode ocorrer a falha do recurso (resultando em novo escalonamento para a mesma tarefa). Em [Silva et al. 2003] é mostrado que através da submissão de réplicas das tarefas pode-se obter um desempenho comparável a soluções dependentes de conhecimento prévio sobre os recursos e tarefas. Uma solução ainda mais aprimorada que utiliza fila de tarefas, replicação e *checkpointing* é proposta em [Anglano and Canonico 2005].

Em [Hurfin et al. 2003] e [Hurfin et al. 2006] é apresentada uma abordagem de escalonamento baseada em leilão, com acordo entre os participantes. Basicamente, os recursos são organizados em domínios, seguindo uma hierarquia multi-nível. O leilão é realizado para determinar como serão distribuídas as tarefas, tanto globalmente no nível da grade quanto localmente nos domínios. Esses leilões dependem de estimativas sobre os tempos de execução das tarefas. Uma questão importante é a dificuldade de se obter tais estimativas de tempo em situações reais. De qualquer forma, a abordagem apresentada é interessante, embora a latência possa causar problemas no caso de uma grade com muitos domínios. Uma análise mais detalhada sobre algoritmos de escalonamento em grades pode ser encontrado em [Dong and Akl 2006].

4. Algoritmos de Escalonamento em Grades

Neste trabalho são propostas quatro alternativas de algoritmos de escalonamento em grades, que são descritas a seguir. Todos eles baseiam-se no princípio de que cada escalonador (processo que realiza o escalonamento) mantém uma visão dos recursos da grade. Tal visão contém informações como disponibilidade, estado das tarefas alocadas e capacidade de processamento. É importante ressaltar que quanto maior for a quantidade de escalonadores, mais complexa é a manutenção da uniformidade das visões. A diferença entre os algoritmos está na maneira como é realizado o mapeamento com o uso das visões, como será detalhado adiante.

Para estimar qual recurso estaria melhor habilitado para resolver uma tarefa, foi

determinada a métrica *Rank* (equação 1), que estabelece a relação entre a quantidade de tarefas alocadas por recurso. Foi determinado que quanto menor o *Rank*, melhor habilitado estaria o recurso. Essa heurística é utilizada em todos os algoritmos propostos neste trabalho. Em todos os casos, informações sobre as tarefas não precisam ser fornecidas, como o tempo médio de execução da tarefa no recurso ou o tamanho da tarefa (medido, por exemplo, pela quantidade de instruções).

$$Rank = Velocidade\ do\ recurso\ em\ MIPS \times \frac{Tarefas\ Alocadas}{Processadores} \quad (1)$$

4.1. Abordagem 1: Escalonadores com visão individual dos recursos da grade

Neste algoritmo, cada escalonador mantém sua visão dos recursos baseada nas tarefas que ele mesmo submete. Assim, o escalonamento é independente da atividade dos demais, e tarefas são alocadas na medida em que chegam. A cada alocação e liberação, a visão é atualizada individualmente. Como os escalonadores utilizam a mesma heurística, pode ocorrer o envio de várias tarefas a um mesmo recurso, resultando em maior utilização de determinados recursos, filas de espera e maior tráfego na rede. Esta é naturalmente a abordagem com implementação mais simples.

Este algoritmo funciona de maneira simples e assemelha-se às soluções baseadas em fila de tarefas, como em [Silva et al. 2003]. No caso, as soluções diferem-se porque [Silva et al. 2003] utiliza replicação e não depende de nenhuma informação sobre o recurso. O algoritmo aqui utilizado não adota o uso da replicação por não sugerir melhora nos índices considerados nos experimentos realizados.

4.2. Abordagem 2: Escalonadores com visão individual atualizada dos recursos da grade

Neste algoritmo, cada escalonador mantém uma visão dos recursos baseada nas tarefas que ele mesmo submete. Entretanto, tal visão é atualizada em certos períodos através da comunicação com um gerenciador de recursos (como o GIS). O escalonamento ocorre de maneira análoga a anterior, porém, cada alocação e liberação deve ser informada ao gerenciador de recursos. Esta abordagem proporciona uma melhor distribuição das tarefas nos recursos, mas exige uma comunicação confiável entre o escalonador e o gerenciador de recursos; caso contrário, visões inconsistentes dos recursos podem ser geradas e mantidas a longo prazo.

4.3. Abordagem 3: Escalonadores com visão compartilhada dos recursos da grade

Neste algoritmo, a visão de cada escalonador é oferecida pelo gerenciador de recursos. Assim, todos os escalonadores possuem visão única, atualizada em tempo real, caracterizando um escalonador único e universal. De fato, cada escalonador individual envia suas tarefas para o escalonador universal. Na prática, esta implementação poderia levar a uma centralização do processo de escalonamento, concentrando o tráfego de rede num único ponto e levando a maiores prejuízos no caso de uma falha no gerenciador. Será considerado como um caso ideal, por se comportar como um escalonador global, dinâmico e ótimo [Dong and Akl 2006]. Observa-se que, em ambiente real, uma solução deste tipo seria completamente inviável devido a problemas de escalabilidade. De fato, soluções semelhantes podem ser obtidas por heurísticas ou aproximação, como descrito em [Gehring and Preiss 1999].

4.4. Abordagem 4: Escalonadores com visão individual e escalonamento baseado em consenso

Neste algoritmo, existe uma visão individual dos recursos; porém, o escalonamento de tarefas é baseado em consenso entre os escalonadores. Periodicamente, é realizado um consenso para decidir como serão alocadas as tarefas recebidas por todos os participantes. Logo, cada escalonador de alto nível tem o conhecimento global das tarefas alocadas pelos demais a cada decisão obtida através do consenso. Desta forma, o consenso oferece uma visão global comum do escalonamento a partir da visão individual de cada escalonador. O interessante é que tal visão comum é acordada periodicamente durante o escalonamento e não precisa ser atualizada globalmente. Cada processo atualiza apenas a sua visão local, considerando-se apenas as suas tarefas e estado local.

A Figura 1 apresenta um diagrama simplificado do algoritmo. Um escalonador (em estado livre) inicia o processo enviando um convite para os demais (passo 1). Em seguida, todos enviam por difusão (*broadcast*) as informações necessárias, isto é, a visão de cada um e sua lista de tarefas (passo 2). Recebidas as informações de todos os escalonadores (passo 3), cada um monta seu esquema de escalonamento e envia para todos os demais, caracterizando um voto (passo 4). Cada participante deve aguardar os votos dos demais (passo 5). O esquema mais votado é adotado (passo 6) e utilizado para a alocação das tarefas, onde cada escalonador aloca apenas suas tarefas, mas conhece as tarefas dos concorrentes.

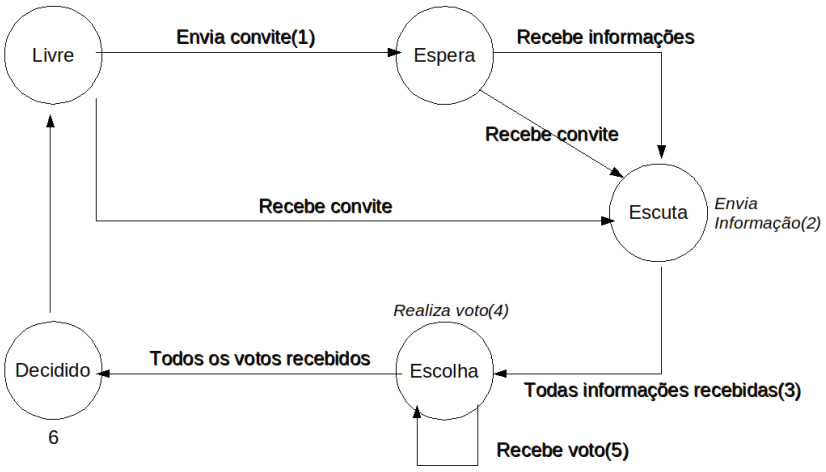


Figura 1. Diagrama de estados do scheduler durante o consenso

O algoritmo aqui apresentado propõe uma melhoria em relação ao proposto por [Hurfin et al. 2003]. Esta melhoria refere-se ao conjunto de participantes do consenso. Em [Hurfin et al. 2003], a quantidade de domínios (ou de recursos) é um fator crucial no funcionamento da grade. Caso haja vários domínios, geograficamente dispersos, o tempo gasto para realizar o consenso pode tornar a solução impraticável (devido sobretudo ao caráter dinâmico do sistema). Por outro lado, limitar o leilão a uma certa quantidade de

domínios também limita a capacidade de processamento da grade. Na abordagem aqui proposta, os participantes do consenso são os próprios escalonadores. Desta forma, a quantidade de recursos não afeta o funcionamento de forma tão crítica.

5. Testes e Avaliação de Desempenho

Cenário dos Testes. Foram realizados testes, a partir de simulações, com o objetivo de analisar o comportamento dos algoritmos propostos neste trabalho. Os testes realizados foram baseados em um arquivo *log* de uma grade real [Feitelson 2005]. Foram extraídas informações sobre 190 recursos da grade com suas respectivas cargas de trabalho, incluindo os tempos de chegada das tarefas. Em termos de comunicação, foram adotadas velocidades de transmissão da ordem de $200MBytes/sec$, enquanto o tamanho médio das tarefas foi ajustado para $300Bytes$. A velocidade dos processadores foi estimada baseando-se no tempo médio de execução de cada tarefa. O período para atualização das visões foi ajustado para 2 minutos.

Para as simulações foi utilizado o *GridSim* [Buyya and Murshed 2002]. Realizou-se simulações envolvendo todos os recursos e suas respectivas cargas de trabalho, composta de 188.041 tarefas independentes. As tarefas foram distribuídas entre os escalonadores de maneira uniforme. Foram realizados testes com escalonadores trabalhando paralelamente, numa variação de 5 a 50 escalonadores.

5.1. Comparação entre as Abordagens 1 e 2

Este caso de teste teve como objetivo avaliar o impacto da atualização das informações sobre o desempenho, considerando a concorrência no uso dos recursos. Os resultados obtidos encontram-se na Tabela 1. Conforme pode ser observado, o aumento na quantidade de escalonadores ($Qtde$) foi melhor aceito pela Abordagem 2, se comparada a Abordagem 1.

Tempo de espera na fila do recurso (seg)						
	Abordagem 1			Abordagem 2		
$Qtde$	5	10	50	5	10	50
Média	52,83	138,59	1006,01	0,20	0,48	0,74
Desvio	462,90	1049,42	5801,45	37,83	72,27	70,29

Tabela 1: Comparação entre os tempos de espera em fila das Abordagens 1 e 2

Cada recurso corresponde a um grupo de processadores gerenciados por um escalonador local. Nas grades esses recursos podem ser *Clusters*, NoWs, MPPs ou supercomputadores. Também foi comparada a quantidade de processadores ocupados de toda a grade, segundo a equação *Processadores Ocupados*2. Seja *Qtde_Rec*, a quantidade de recursos da grade, *Proc_Ocup*, a quantidade de processadores ocupados do recurso e *Total_Proc*, o total de processadores do recurso. Então:

$$Processadores\ Ocupados = \sum_{Qtde\_Rec} \frac{Proc\_Ocup}{Total\_Proc} \quad (2)$$

Os resultados obtidos são apresentados nas Figuras 2, 3 e 4 (quantidade de processadores ocupados em função do tempo) e mostram que a atualização das visões causa um menor número de processadores ocupados praticamente a todo tempo. Quanto menos processadores ocupados, menor a possibilidade de se ter uma fila de espera para o recurso. Neste caso, pelos testes efetuados, verifica-se que a Abordagem 2 demonstrou um desempenho superior.

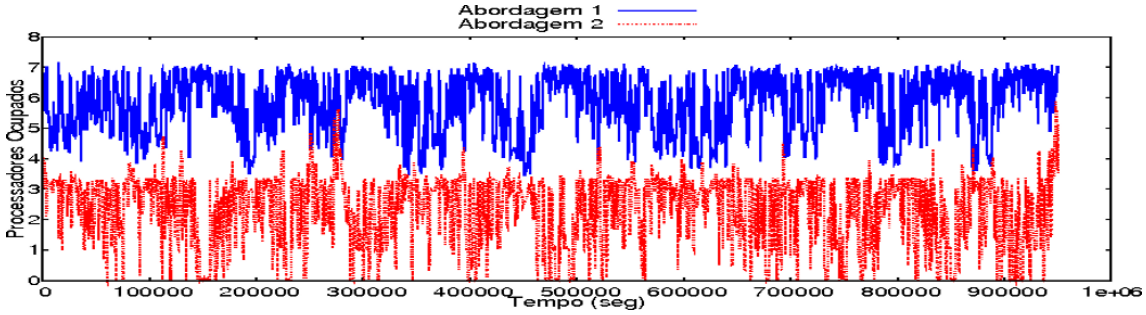


Figura 2. Processadores ocupados, 5 escalonadores

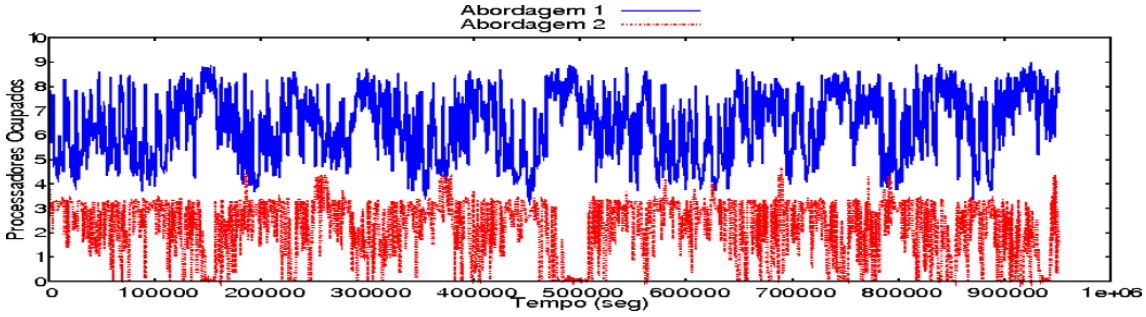


Figura 3. Processadores ocupados, 10 escalonadores

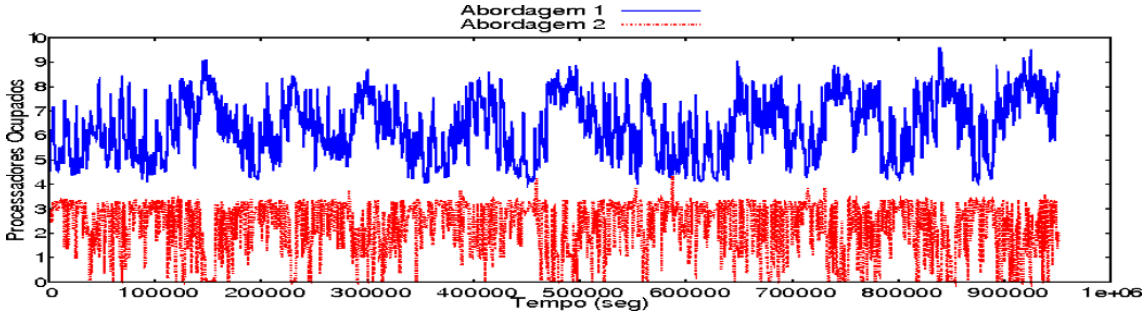


Figura 4. Processadores ocupados, 50 escalonadores

5.2. Comparação entre as Abordagens 3 e 4

Foram realizadas simulações a fim de se comparar os algoritmos cooperativo (Abordagem 3, Seção 4.3) e baseado em consenso (Abordagem 4, Seção 4.4). Os resultados obtidos encontram-se na Tabela 2. Quanto mais escalonadores, maior o tempo necessário para o consenso. O desempenho do sistema é afetado, pois tarefas podem ter sido liberadas ou alocadas durante a troca de mensagens do consenso, gerando decis oes baseadas em informações decorrentes de visões desatualizadas e, consequentemente, filas nos recursos.

Na Abordagem 3, o escalonador universal aloca as tarefas da melhor maneira possível (considerando as métricas estipuladas). Devido a quantidade de processadores dos recursos, não há tempo de espera em nenhum caso.

Tempo de espera na fila do recurso (seg)				
	Abordagem 4		Abordagem 3	
$Qtde$	5	10	5	10
Média	153,80	452,80	0,0	0,0
Desvio	1203,50	3468,69	0,0	0,0

Tabela 2: Comparação entre os tempos de espera em fila das Abordagens 3 e 4

A Tabela 3 mostra estatísticas sobre a ocupação do sistema segundo a equação de *Processadores Ocupados* (equação 2). Naturalmente, a abordagem cooperativa obteve melhor resultado, por perseguir o mesmo comportamento de um escalonador global. A diferença foi causada pelo uso de visões desatualizadas, geradas pelas alocações e liberações ocorridas durante o consenso. Quanto mais tarefas, mais acentuado o efeito da desatualização das informações.

Ocupação do Sistema				
	Abordagem 4		Abordagem 3	
$Qtde$	5	10	5	10
Média	2,55	6,56	2,57	2,55
Desvio	0,58	1,21	0,58	0,59
Máximo	3,32	9,03	3,33	3,34

Tabela 3: Comparação entre as médias de ocupação do sistema das Abordagens 3 e 4

5.3. Comportamento do Algoritmo Baseado em Consenso (Abordagem 4)

Foram realizadas simulações para avaliar o comportamento do escalonador baseado em consenso. Nestas medidas, foi analisado os tempos de espera com a variação da quantidade de recursos (5, 10 e 20) e a quantidade de escalonadores (5, 10, 15, 20, 25).

As Tabelas 4, 5 e 6 mostram o tempo decorrido (uma média de todos os escalonadores) entre a submissão e a alocação da tarefa (latência), em função da quantidade de escalonadores ($Qtde$). Pode ser observado que muito tempo é gasto neste processo, em parte decorrente do fluxo de comunicação (proporcional a quantidade de escalonadores). De fato, sempre será necessário tempo adicional para a realização do consenso. Isso pode se agravar caso a quantidade de participantes seja muito grande, uma vez que todos trocam mensagens entre si e essas mensagens devem chegar na ordem correta. Dessa forma, o escalonamento pode gerar grande latência devido ao próprio consenso.

Tempo decorrido da chegada da tarefa até a sua alocação (seg)					
$Qtde$	5	10	15	20	25
Média	484,63	699,74	877,30	1129,75	1125,87
Desvio	1048,52	1329,00	1761,77	2427,78	1940,89

Tabela 4: Latência, para 5 recursos e 2694 tarefas

Tempo decorrido da chegada da tarefa até a sua alocação (seg)					
$Qtde$	5	10	15	20	25
Média	154,71	208,25	235,99	272,56	235,99
Desvio	227,14	311,01	344,82	406,41	344,82

Tabela 5: Latência, para 10 recursos e 9862 tarefas

Tempo decorrido da chegada da tarefa até a sua alocação (seg)					
$Qtde$	5	10	15	20	25
Média	98,94	115,93	127,85	288,17	140,10
Desvio	93,67	117,31	131,35	417,98	150,36

Tabela 6: Latência, para 20 recursos e 27409 tarefas

5.4. Comparação Geral das Abordagens

Nestes testes foram utilizados todos os recursos e toda a carga de trabalho, contabilizando 190 recursos e 188.041 tarefas, distribuídas de maneira uniforme entre os escalonadores.

A Tabela 7 mostra os tempos de execução obtidos ao utilizar as Abordagens 2, 3 e 4. Obviamente, a Abordagem 3 (cooperativa) comporta-se da mesma maneira quando apenas variamos a quantidade de escalonadores ($Qtde$), devido ao comportamento de escalonador universal. A Abordagem 4 (consenso) é mais sensível à variação da quantidade de escalonadores, devido ao tempo gasto pelo próprio consenso. A atualização periódica da visão individual, utilizada na Abordagem 2, tem um desempenho mais estável.

Tempo de serviço (seg)						
	Abordagem 2		Abordagem 3		Abordagem 4	
$Qtde$	5	10	5	10	5	10
Média	208,28	333,44	51,72	53,73	387,57	730,968
Desvio	628,51	1201,33	187,39	197,42	1355,57	3601,57

Tabela 7: Comparação entre os tempos de execução obtidos nas Abordagens 2, 3 e 4

6. Conclusão e Perspectivas

Este trabalho propôs e analisou o comportamento de alguns algoritmos baseados no princípio de utilizar visões dos recursos na realização do escalonamento de tarefas em grades. Cada algoritmo determina uma maneira de manter e atualizar as informações sobre os recursos. O foco principal deste trabalho foi verificar o desempenho do algoritmo baseado em consenso para atualização dessas informações. Segundo os resultados, o consenso não apresentou grandes melhorias em relação aos tempo de serviço no cenário apresentado, se comparado com a utilização de uma visão individual atualizada.

Utilizar-se de consenso para o escalonamento de tarefas no grade traria resultados melhores caso fosse utilizado um canal de comunicação mais rápido. O tempo de serviço das tarefas tende a aumentar devido ao aumento de tráfego causado pelas mensagens necessárias ao consenso e, conseqüentemente, pelo acréscimo do tempo de espera. É importante citar que o consenso traz outras vantagens não consideradas neste estudo, sobretudo a possibilidade da realocação das tarefas pertencentes a um escalonador que apresentou falha numa posterior rodada de consenso. Para os algoritmos com visão individual, recuperar-se de falhas deste tipo fica mais complicado, devido a falta de informações globais (em outras palavras, uma tarefa perdida na falha de um escalonador não pode ser recuperada porque os demais não sabem da sua existência).

Como trabalhos futuros imediatos, pretende-se avaliar efetivamente o impacto das falhas no processo de escalonamento utilizado pelas abordagens sugeridas. Acreditamos, que em alguns cenários, a abordagem de cálculo de visões a partir do consenso possa ter algumas vantagens.

Vale ressaltar que, apesar de apresentar vantagens na ocorrência de falhas, a abordagem do uso do consenso pode não compensar o aumento do tempo de serviço, uma vez que o problema da falha poder ser contornado pela utilização de replicação das tarefas [Silva et al. 2003] ou técnicas mais avançadas, como *checkpointing* [Jankowski et al. 2005]. Um estudo comparativo entre as vantagens e desvantagens do uso de tais abordagens de tolerância a falhas (consenso, replicação e *checkpointing*), tanto do ponto de vista analítico como experimental, é extremamente pertinente e poderá ser objeto de trabalhos futuros.

Referências

Anglano, C. and Canonico, M. (2005). Fault-tolerant scheduling for bag-of-tasks grid applications. In *Proceedings of the 2005 European Grid Conference (EuroGrid 2005). Lecture Notes in Computer Science*, page 630. Springer.

Buyya, R. and Murshed, M. (2002). Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220.

Chandra, T. D., Hadzilacos, V., and Toueg, S. (1996). The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267.

Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C. (2001). Grid information services for distributed resource sharing. pages 181–194.

Dong, F. and Akl, S. G. (2006). Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 504.

Feitelson, D. (2005). Parallel workloads archive: Lcg grid `http://www.cs.huji.ac.il/labs/parallel/workload/l_lcg/index.html`.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382.

Foster, I. and Kesselman, C. (1998). *The Grid: Blueprint for a New Computing Infraestructure*. Morgan Kaufmann Publishers.

Gehring, J. and Preiss, T. (1999). Scheduling a metacomputer with uncooperative subschedulers. In *Proceedings of IPPS Workshop on Job Scheduling Strategies for Parallel Processing*, pages 179–201. Springer-Verlag.

Greve, F. and Narzul, J. L. (2004). Designing a configurable group service with agreement components. In *Workshop de Testes e Tolerancia a Falhas (WTF 2004), SBRC*, Gramado, Brasil.

Greve, F. G. P. (2005). Protocolos Fundamentais para o Desenvolvimento de Aplicações Robustas. In *Minicursos SBRC 2005: Brazilian Symposium on Computer Networks*, pages 330–398. Sociedade Brasileira de Computação, Fortaleza, CE, Brazil.

Hurfin, M., Le Narzul, J.-P., Pley, J., and Parvedy, P. R. (2006). Paradis: an adaptive middleware for dynamic task allocation in a grid. In *XXIV Simposio Brasileiro de Redes de Computadores (SBRC 2006)*, Curitiba, Brazil.

Hurfin, M., Narzul, J. L., Pley, J., and Parvedy, P. R. (2003). A fault-tolerant protocol for ressource allocation in a grid dedicated to genomic applications. In *Proc. of the Fifth International Conference on Parallel Processing and Applied Mathematics, Special Session on Parallel and Distributed Bioinformatic Applications (PPAM-03)*, LNCS 3019, pages 1154–1161, Czestochowa, Poland. Springer-Verlag.

Jankowski, G., Kovács, J., Meyer, N., Januszewski, R., and Mikolajczak, R. (2005). Towards checkpointing grid architecture. In Wyrzykowski, R., Dongarra, J., Meyer, N., and Wasniewski, J., editors, *Parallel Processing and Applied Mathematics, 6th International Conference, PPAM 2005, Poznan, Poland, September 11-14, 2005, Revised Selected Papers*, volume 3911 of *Lecture Notes in Computer Science*, pages 659–666. Springer.

Krauter, K., Krauter, K., and It, M. M. (2002). A taxonomy and survey of grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32(2):135–164.

Laure, E. and Jones, B. (2008). Enabling grids for e-science: The egee project. Technical Report EGEE-PUB-2009-001. 1.

Schopf, J. M. (2001). Ten actions when superscheduling. `http://http://www.ggf.org/documents/GFD/GFD-I.4.pdf`.

Silva, D. P. D., Cirne, W., Brasileiro, F. V., and Grande, C. (2003). Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In *Applications on Computational Grids, in Proc of Euro-Par 2003*, pages 169–180.

Controle distribuído em Redes de Data Center baseado em Gerenciadores de Rack e Serviços de Diretório/Topologia

Carlos A. B. Macapuna, Christian Esteve Rothenberg, Maurício F. Magalhães

[1]Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101 – 13083-970 – Campinas – SP – Brasil

{macapuna, chesteve, mauricio}@dca.fee.unicamp.br

Abstract. *In this paper, we present and evaluate the design and implementation of two distributed, fault-tolerant services that provide the directory and topology information required to encode randomized source routes with in-packet Bloom filters. By deploying an army of Rack Managers acting as OpenFlow controllers, the proposed architecture promises scalability, performance and fault-tolerance. We show that packet forwarding itself may become a cloud internal service implemented by leveraging cloud application best practices such as distributed key-value storage systems. Moreover, we contribute to demystifying the argument that the centralized controller model of OpenFlow networks is prone to a single point of failure and show that direct network controllers can be physically distributed, yielding thereby a sweet intermediate approach to networking between fully distributed and centralized.*

Resumo. *Neste trabalho, apresentamos e avaliamos a concepção e implementação de dois serviços distribuídos e tolerantes a falhas que fornecem as informações de diretório e topologia necessárias para codificar aleatoriamente rotas na origem usando filtros de Bloom no cabeçalho dos pacotes. Ao implantar um exército de gerenciadores de Rack atuando como controladores OpenFlow, a arquitetura proposta promete escalabilidade, desempenho e tolerância a falhas. O artigo mostra que o encaminhamento de pacotes pode tornar-se um serviço interno na nuvem e que a sua implementação pode aproveitar as melhores práticas das aplicações em nuvem como, por exemplo, o armazenamento distribuído do par <chave,valor>. Além disso, o artigo contrapõe-se ao argumento de que o modelo de comando centralizado de redes (OpenFlow) está vinculado a um único ponto de falha. Isto é obtido através da proposta de uma arquitetura de controle fisicamente distribuída, mas baseada em uma visão centralizada da rede gerando, desta forma, uma abordagem de controle de rede intermediária, entre totalmente distribuída e centralizada.*

1. Introdução

Como as grades computacionais, a nuvem é um modelo de *utility computing*. Este tipo de computação envolve uma dinâmica no aumento e na redução dos recursos os quais, quando agregados, são apresentados aos clientes como um conjunto único de recursos de processamento e armazenamento, virtualmente "infinitos". Embora as aplicações em

nuvem apresentadas para os usuários finais sejam interativas (por exemplo, buscas, E-mail, armazenamento de fotos), estas não seriam possíveis sem as tarefas de análise, processamento e armazenamento de dados em grande escala (por exemplo, MapReduce [Dean and Ghemawat 2010], Hadoop [Wang et al. 2009]). Um suporte adequado para esse conjunto de serviços heterogêneos na mesma infraestrutura impõe requisitos de rede tais como: (i) vazão e latência uniformes entre qualquer par de servidores para matrizes de tráfego altamente variáveis e dominadas por rajadas; (ii) suporte à mobilidade sem interrupção de cargas de trabalho; e (iii) a alocação ágil de máquinas virtuais em qualquer servidor físico disponível. Estes e outros requisitos da infraestrutura de TI impostos pelo modelo de *cloud computing* têm motivado a pesquisa de novos projetos e arquiteturas de rede adequadas para os *cloud data centers* de próxima geração [Verdi et al. 2010].

Com o surgimento de switches programáveis baseados na tecnologia OpenFlow [McKeown et al. 2008], emerge um promissor conceito de redes controladas via *software* ou, em inglês, *software-defined networking* [Greene 2009]. O protocolo OpenFlow especifica um caminho padrão para controle das tomadas de decisões no tratamento de pacote (por exemplo, encaminhar na porta x, descartar, enviar para o controlador, etc.) através de uma inteligência (*software*) implementada em controladores logicamente centralizados, mantendo os fornecedores de *hardware* responsáveis apenas pelo desenvolvimento do dispositivo (com suporte à tabela de fluxos especificada pelo OpenFlow). Neste modelo de rede, a abordagem tradicional de gerência de baixo nível (por exemplo, endereços MAC e IP) e a operação de protocolos e algoritmos distribuídos (por exemplo, Bellman-Ford), tornam-se um problema "basicamente" de *software*, com visão global da rede (por exemplo, Dijkstra) e um maior nível de abstração (por exemplo, nomes de usuário, FQDN, etc.). Dessa forma, a comunidade de desenvolvedores de sistemas distribuídos que tem contribuído com a realização dos denominados *mega data centers* ou *warehouse-scale computers* [Barroso and Hölzle 2009], pode definir o comportamento e novas funcionalidades da rede conforme os objetivos e requisitos específicos do provedor da infraestrutura e serviços, sem ter que depender dos demorados ciclos de atualização dos equipamentos de rede. Por isso, não é surpresa que provedores de infraestrutura para *cloud data centers*, tais como Google ou Amazon, analisem com grande interesse a tecnologia OpenFlow. A grande razão para este interesse reside nos níveis de controle e flexibilidade que eles precisam na provisão dos serviços na nuvem a um custo inferior mas com o mesmo desempenho dos switches *comoditizados* atualmente em operação.

Motivados por esse cenário de desafios e oportunidades, projetamos e implementamos em trabalho anterior [Rothenberg et al. 2010] uma arquitetura de rede de centro de dados (DCN - *data center networks*) que oferece um novo serviço de encaminhamento baseado na codificação de rotas na origem em um filtro de Bloom nos pacotes (iBF - *in-packet Bloom filter*), no caso, adicionado aos campos de endereço do quadro MAC/Ethernet. Neste trabalho, estendemos a proposta para um ambiente de implementação totalmente distribuído. Nesta nova arquitetura são introduzidos dois serviços, escaláveis e tolerantes a falhas, para manter globalmente apenas as informações de topologia e diretório de servidores. Estas informações são mantidas em um sistema de armazenamento distribuído <chave,valor>. Com o desafio de prover uma arquitetura resiliente, o controle centralizado do gerenciador de *rack* (RM - *Rack Manager*) introduzido na nossa arquitetura denominada SiBF [Rothenberg et al. 2010] passa a ser um "exército" de *Rack Managers* com uma instância executando em cada *rack*. Esta abordagem ofe-

rece escalabilidade, desempenho e tolerância a falhas, aproveitando a disponibilidade e as características de bases de dados não relacionais tipicamente disponíveis em *data centers* como serviço de suporte ao desenvolvimento das aplicações em nuvem.

Os resultados deste trabalho sugerem que, suportado pela disponibilidade de switches programáveis de prateleira (*commodity*), um serviço típico de rede como, por exemplo, o encaminhamento de pacotes, pode tornar-se mais um serviço da nuvem privada, provendo uma interligação eficiente dos servidores de aplicações e implementado segundo as boas práticas do desenvolvimento de aplicações distribuídas nos *cloud data centers* (baixa latência, confiabilidade e escalabilidade). Além disso, o trabalho questiona o argumento de que o modelo de controle centralizado, como nas redes OpenFlow, possui um único ponto de falha. O artigo mostra que os controladores de rede podem ser fisicamente distribuídos mas com uma visão centralizada dos recursos da rede, resultando em uma abordagem de rede intermediária, entre totalmente distribuída e centralizada.

O restante deste trabalho está estruturado conforme descrito a seguir. A Seção 2 apresenta informações relacionadas à arquitetura proposta, assim como, seus princípios. Na Seção 3 são detalhados os principais pontos correspondentes à implementação do protótipo. Na Seção 4 a proposta é avaliada do ponto de vista da tolerância a falhas. Finalmente, na Seção 6, são apresentadas as conclusões e os trabalhos futuros.

2. Arquitetura distribuída proposta

A proposta fundamenta-se na centralização da inteligência de controle da rede (conforme modelo 4D [Greenberg et al. 2005] - *decision*, *dissemination*, *data*, *discovery*) através da remoção da operação distribuída presente em vários protocolos como, por exemplo, a resolução de endereço realizada pelo protocolo ARP. A tomada de decisão é desviada para um elemento controlador e o plano de dados (encaminhamento) é realizado pelos switches. A seguir, descrevemos a topologia, os princípios de projeto e os principais blocos funcionais da nossa proposta.

2.1. Topologia

A distribuição dos equipamentos de rede em um *cloud data center* é tipicamente organizada de acordo com uma topologia *fat-tree*. A opção por esta topologia deve-se à facilidade de implementação através de switches baratos e largamente disponíveis (*commodities*) e por permitir um melhor uso dos recursos da rede [Al-Fares et al. 2008]. A topologia *fat-tree* mostrada na Figura 1 utiliza três camadas: uma camada inferior formada por switches ToR (*Top of Rack*); uma camada intermediária constituída por switches de agregação (Aggr); e uma camada superior de switches de núcleo (Core). Esta abordagem proporciona múltiplos caminhos entre qualquer par de servidores, além de facilitar o balanceamento de carga e a tolerância a falhas. Os nós finais conectados aos ToRs podem ser qualquer servidor de aplicação (físico ou virtual), inclusive os que oferecem serviços para suportar o funcionamento da infraestrutura como, por exemplo, o *Rack Manager* e o sistema de armazenamento de dados.

2.2. Princípios de projeto

As redes de *data center*, comparativamente à Internet, possuem uma escala muito menor e modelos de controle e gerência que facilitam a adoção de novos paradigmas. Baseado nestas características, os princípios de projeto são relacionados a seguir:

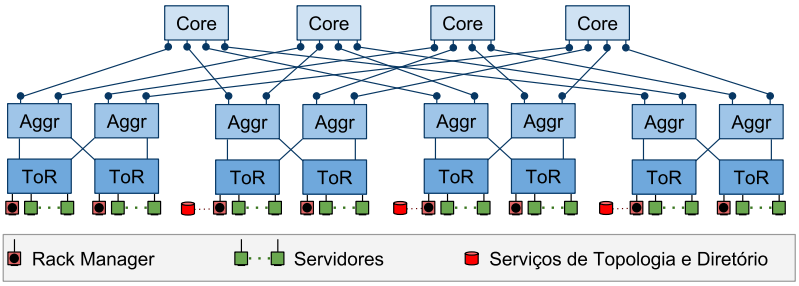


Figure 1. Topologia fat-tree de três camadas.

Separação identificador/localizador: A divisão entre identificador e localizador possui um papel fundamental para viabilizar o compartilhamento dos serviços baseados no endereçamento IP. O IP é utilizado apenas para identificar servidores físicos e as máquinas virtuais (VMs - *virtual machines*) dentro do *data center*. Desta forma, diferentemente da atribuição de endereços IP baseada na hierarquia de provedores adotada na Internet, não são impostas restrições de como os endereços são atribuídos ou utilizados para acesso externo (clientes espalhados na Internet). Esta opção torna o endereço IP não significativo para o encaminhamento de pacotes dentro da infraestrutura do *data center*. Este encaminhamento é realizado no nível da camada 2, modificada para oferecer um serviço de rota na origem de forma transparente aos nós e aplicações legadas.

Rota na origem: De modo a aproveitar o pequeno diâmetro das topologias de redes de *data center*, a abordagem adotada pela arquitetura utiliza o esquema de rota na origem (*strict source routing*). O encaminhamento nas topologias de redes de *data center* em 3 camadas (Figura 1) é bastante simplificado, ou seja, qualquer rota entre dois ToRs tem uma trajetória ascendente em direção a um switch Core e, em seguida, uma trajetória descendente em direção ao ToR destino, ambas passando por switches intermediários (Aggr). Esta abordagem permite enviar pacotes por rotas diferentes (balanceamento de carga), além de permitir a seleção da rota menos congestionada (engenharia de tráfego). A especificação da rota na origem é baseada na utilização do filtro de Bloom nos pacotes (iBF - *in-packet Bloom filter*) contendo apenas três elementos identificadores de switches $\langle Core, Aggr_{desce}, ToR_{destino} \rangle$. O identificador utilizado neste trabalho corresponde ao endereço MAC do switch. O esquema básico adotado para operacionalização do filtro de Bloom consiste na programação do ToR para adicionar, nos campos src-MAC e dst-MAC do cabeçalho do quadro Ethernet, o filtro de Bloom contendo a rota. Na sequência, o ToR encaminha o quadro para o Aggr de subida. Deve ser ressaltado que não há necessidade de incluir o identificador do Aggr de subida no filtro de Bloom já que esta informação é implícita ao ToR. Nos próximos saltos, apenas três encaminhamentos são realizados utilizando o iBF, conforme exemplificado na Figura 2.

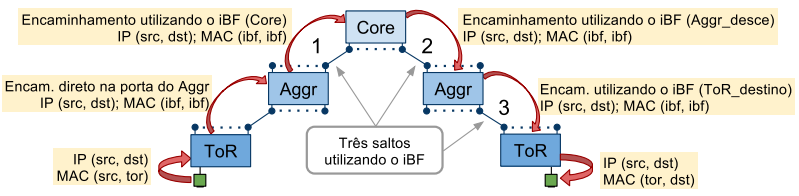


Figure 2. Encaminhamento utilizando iBF.

Controle logicamente centralizado e fisicamente distribuído: Nossa abordagem adota a proposta incluída em 4D [Greenberg et al. 2005] e que sugere a separação da tomada de decisão (controle) do encaminhamento (dados) e introduz um novo elemento de rede, o controlador. Porém, adotamos, o que denominamos de abordagem de rede intermediária, ou seja, um controle logicamente centralizado, onde apenas o estado da topologia e do diretório são mantidos globalmente, mas com controladores distribuídos para atuação sobre um número limitado de switches. Esses controladores são o que chamamos de *Rack Managers* e controlam os switches OpenFlow.

Balanceamento de carga: Para fornecer uma distribuição do tráfego na rede para qualquer matriz de tráfego, a abordagem adotada é o espalhamento dos fluxos por caminhos aleatórios (*oblivious routing* [Yuan et al. 2007]). O *Rack Manager* é responsável pela seleção aleatória das rotas compactadas nos iBF na direção do ToR de destino. A estratégia de balanceamento adotada pela arquitetura proposta neste artigo baseia-se no *valiant load balancing* (VLB como usado em [Greenberg et al. 2009]) que roteia de forma aleatória os fluxos na camada de switches intermediários (Aggr e Core).

Tolerância ampla à ocorrência de falhas: A premissa de que qualquer elemento pode falhar é adotada na concepção e implementação da arquitetura. Em uma infraestrutura com dezenas de milhares de elementos, falhas são comuns e constantes. Considerando um *data center* com 30.000 elementos (servidores e switches), há de se esperar que todo dia tenha, pelo menos, um elemento com mau funcionamento. Neste contexto, o desenho da arquitetura e os serviços de rede devem ser à prova de falhas (*design for failure*), assumindo que qualquer componente pode falhar a qualquer momento. Este princípio (e o controle logicamente centralizado e fisicamente distribuído) é a principal contribuição deste trabalho em relação ao SiBF e por isso, a avaliação será feita sobre tolerância a falhas, os demais já foram discutidos em [Rothenberg et al. 2010].

2.3. Serviço de encaminhamento livre de falsos positivos

Devido às suas propriedades, a adoção de filtros de Bloom para o encaminhamento de pacotes pode ocasionar falsos positivos com uma certa probabilidade.[1] A seguir, são apresentados, de forma resumida, o problema e a solução adotada sem falsos positivos.

Filtro de Bloom: Um filtro de Bloom é uma estrutura de dados que identifica se determinado elemento está ou não presente nessa estrutura. É implementado como um vetor de m bits, com n elementos inseridos por k funções de *hash* independentes. Cada elemento, ao ser inserido no filtro, passa por uma função de *hash* cujo ação consiste na atribuição do valor 1 ao bit na posição do vetor associada ao resultado do *hashing*. Para verificar se algum elemento está presente, os resultados de todas as k funções de *hash* devem possuir o valor 1 no vetor. Basta um único dos bits anteriores apresentar o valor zero para concluir que o elemento não está presente. Neste caso, não há possibilidade da ocorrência de falsos negativos. Um falso positivo ocorre quando todos os resultados das k funções de *hash* em um elemento que não foi inserido apresentam o valor 1.

Impacto de falsos positivos no encaminhamento: O serviço de topologia, após a descoberta da topologia, instala na tabela de encaminhamento do switch uma entrada de fluxo correspondente à cada porta de saída do switch. Cada uma destas entradas corresponde a um filtro de Bloom codificado com a identificação do vizinho detectado. Quando

[1]Mais detalhes do tratamento dos falsos positivos nos iBFs, encontram-se em [Rothenberg et al. 2010].

um pacote com iBF chega ao switch, a máscara de bits é comparada com as entradas na tabela; se uma entrada for encontrada, o pacote é enviado à porta correspondente. Um falso positivo ocasiona o envio do pacote para duas (ou mais) interfaces de saída.

Removendo os falsos positivos: Uma opção para contornar a ocorrência de falsos positivos é a realização do *multicast*. Neste caso, o pacote será encaminhado pelo caminho certo e por mais uma porta e, no próximo switch, com alta probabilidade, será descartado ao carecer de uma entrada válida.[2] Outra opção consiste no encaminhamento *unicast*. Nesta opção, se a escolha por um dos caminhos não for a opção correta, o pacote será encaminhado pelo próximo switch (que não terá uma entrada válida para encaminhar aquele pacote) ao controlador permitindo, neste caso, que uma nova rota seja definida. No entanto, como o controlador possui a visão global da topologia e existem múltiplos caminhos devido a topologia *fat-tree*, a opção adotada pela arquitetura consiste na eliminação dos caminhos sujeitos a falsos positivos. Desta forma, temos um plano de encaminhamento livre de falsos positivos. As simulações realizadas sobre topologias de grande escala (mais de 500 switches e 10.000 servidores físicos) [Rothenberg et al. 2010] têm demostrado que esta solução resulta na invalidação, ou seja, na não utilização, de menos de 1% dos múltiplos caminhos disponíveis entre quaisquer dois ToRs.

2.4. Serviços de Topologia e Diretório

Assim como outras propostas de novas arquiteturas de *data center* (por exemplo, VL2 [Greenberg et al. 2009] e Portland [Mysore et al. 2009]), a arquitetura apresentada neste trabalho propõe um serviço escalável de diretório para manter o mapeamento entre o identificador IP do servidor (ou máquina virtual) e o switch de borda ToR onde ele está ligado. O serviço de topologia é o resultado de um protocolo de descoberta de topologia baseado em uma extensão do LLDP. Ambos os serviços (topologia e diretório) fornecem as informações globais necessárias às funções de controle. O controle do encaminhamento dos fluxos no *data center* tolera um certo grau de inconsistência nas informações fornecidas pelos serviços de topologia e diretório. Esta flexibilidade permite a implementação destes serviços através de um sistema distribuído escalável, confiável e de alto desempenho, do mesmo modo como em muitas outras aplicações na nuvem.

Base de dados não relacionais: A arquitetura proposta no trabalho adota um sistema de base de dados distribuído baseado no par <chave, valor> do tipo NoSQL (*Not only SQL*) [NoSQL]. As implementações nos *cloud data center* utilizando uma estrutura desse tipo apresentam APIs de fácil utilização e atendem os requisitos desejáveis (consistência, escalabilidade, confiabilidade). Bancos de dados relacionais apresentam custos elevados e não se justificam quando não se requer garantias estritas de *ACID* (atomicidade, consistência, isolamento e durabilidade). As bases de dados NoSQL disponíveis são distribuídas e apresentam, geralmente, uma eficiente replicação dos dados, formando uma base para a implementação de serviços de apoio ao desenvolvimento de aplicações. Tipicamente a API oferece operações simples (`get(key)`, `set(key, value)`, `remove(key)`) e a sua implementação e funcionamento internos (replicação, tolerância a falhas, consistência, versionamento) são transparentes ao desenvolvedor.

Serviço de Topologia: O conhecimento da topologia é um pré-requisito para o encaminhamento com rota na origem. O *Topology Service* (TS) deve manter atualizado o

[2]Porém, em função da topologia, esta abordagem pode ocasionar *loops*.

estado global da topologia da rede procurando oferecer a maior consistência possível para as informações. A descoberta da topologia realiza a identificação dos switches (Core, Aggr e ToR), associa o vizinho descoberto a uma interface de saída (porta) e instala as entradas nos switches Aggr e Core contendo o filtro de Bloom com a identificação do vizinho. As informações da topologia, descobertas através de cada TS, são inseridas na base de dados por quaisquer dos RMs responsável pelo controle dos switches e pela implementação distribuída do protocolo de descoberta. O TS também é responsável pela recuperação da topologia (global) e por disponibilizá-la como um serviço ao RM.

Serviço de Diretório: Com a separação identificador e localizador, o *Rack Manager* necessita conhecer previamente o ToR em que o IP (identificador) de destino está conectado. Como um dos objetivos da arquitetura é a escalabilidade, a propagação de pacotes ARP é eliminada. Portanto, o *Directory Service* (DS) deve manter um mapeamento entre IP e ToR. Esta informação é mesclada na base de dados, e o próprio DS recupera os mapeamentos dos outros ToRs permitindo-o ter uma visão do estado global. Outro mapeamento, que não é requisito de estado global, mas único de cada ToR, é a associação do IP e MAC à porta de saída do ToR. Para recuperação de falhas do RM, esse mapeamento também é armazenando na base de dados.

3. Implementação

Conforme a ilustração dos componentes na Figura 4, a implementação do encaminhamento dos iBF é baseada em switches OpenFlow, enquanto o *Rack Manager* e os *Topology Service* e *Directory Service* são implementados como aplicações adicionadas ao controlador NOX [Gude et al. 2008]. Já a base de dados estende uma implementação do sistema NoSQL Keyspace [Trencseni and Gazso]. A seguir, descrevemos as principais questões relacionadas à implementação do protótipo e do ambiente de testes.

3.1. OpenFlow

A principal característica do OpenFlow (OF) consiste na definição de uma de uma tabela de fluxos cujas entradas contêm um conjunto de campos do cabeçalho de pacote. Este conjunto é composto por uma tupla formada por 10 elementos e por uma lista de ações suportadas via *hardware* como, por exemplo, o envio de um fluxo para uma determinada porta, o encapsulamento e transmissão do pacote para o controlador ou, ainda, o descarte do pacote. Para viabilizar a implementação do protótipo, e a fim de permitir o encaminhamento com base na codificação do filtro de Bloom incluída nos campos de endereço MAC do quadro Ethernet, foi introduzida uma pequena alteração na implementação do OF (v. 0.89rev2 e v 1.0). A Figura 3 mostra a tupla de campos de cabeçalho disponível pelo OF e identifica os campos que são utilizados pela arquitetura, no caso, os dois campos de endereço Ethernet (`src` e `dst`) para inclusão do iBF de 96 bits, e o campo relacionado ao IP, o qual é interpretado na arquitetura como identificador do nó final.

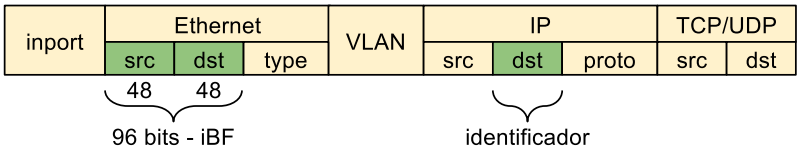


Figure 3. 10 tuplas disponíveis pelo OF e as 3 utilizadas.

3.2. Gerenciador de Rack

O *Rack Manager* (RM) atua como um controlador de switches e a sua implementação é instanciada como um componente que executa no contexto do controlador NOX. A interface de programação do NOX é construída sobre os eventos, seja por componentes principais do NOX (*core*), ou definidos por usuários e gerenciados diretamente a partir de mensagens OF como `packet-in`, `switch join`, `switch leave`. Para realizar o encaminhamento com o iBF e fornecer os serviços de topologia e diretório, foi necessário o desenvolvimento de alguns componentes adicionais para o NOX (ver Figura 4).

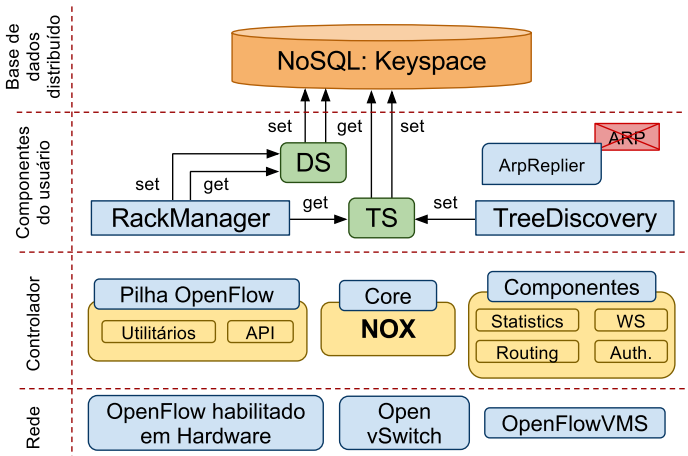


Figure 4. Arquitetura do SiBF.

Rack Manager: É responsável pelo gerenciamento dos novos fluxos que chegam ao switch e a respectiva configuração destes fluxos através dos serviços de diretório e topologia. O *Rack Manager* mantém uma base de dados em *cache* com informações dos servidores descobertos em cada ToR e um mapeamento entre IP, MAC e porta do ToR no qual estes servidores estão anexados. Estas informações são atualizadas no *Directory Service*. O *Rack Manager* interage periodicamente com o *Topology Service* e o *Directory Service* para obter a topologia completa e as informações globais de diretório. Desta forma, o RM constrói uma matriz com os candidatos iBFs livres de falsos positivos para cada caminho entre qualquer combinação de pares de ToRs.

Tree Discovery: Um ponto não trivial é a inferência correta da topologia da árvore e o papel que cada switch desempenha na rede (ou seja, ToR, Core ou Aggr). Para isso, foi desenvolvido um algoritmo que automatiza esta inferência. O *Role Discovery Protocol* utiliza uma extensão do LLDP (*Link Layer Discovery Protocol*) para divulgar a identificação de cada switch. Esta informação é propagada entre os vizinhos, onde cada um descobre qual é o seu papel na topologia (ToR, Aggr ou Core).[3]

Arp Replier: Possui a tarefa de registrar e responder aos pacotes ARP que chegam aos switches de borda, com isso, eliminando a inundação na rede (*flooding*) ocasionada pela propagação de pacotes ARP. Para qualquer requisição de ARP, o *Arp Replier* inclui na resposta, no campo de MAC destino, o endereço MAC associado ao ToR. Isso garante que a tabela ARP do servidor (físico ou virtual) tenha uma entrada, mesmo que única, para cada IP de destino, onde o ToR acaba atuando como se fosse um *gateway* de IP.

[3]Os detalhes do protocolo encontram-se em [Rothenberg et al. 2010].

Topology Service e Directory Service: A implementação do *Directory Service* e *Topology Service* segue uma mesma metodologia. Oferecem métodos de inserção e recuperação das informações na base de dados. Os métodos de inserção recebem apenas uma chave e um valor. Cada método é responsável por codificar a chave em uma outra chave para armazenamento na base de dados, composta pela chave mais um prefixo (`set("pre+key", value)`). Isto é necessário pois os dados encontram-se em um único domínio e, neste caso, o prefixo é utilizado para distinguir os tipos de informações (por exemplo, lista de Aggr, mapeamento IP/ToR). Quando uma informação é solicitada, um conjunto de valores é retornado baseado na chave e no prefixo associados. Esta decisão de implementação deve-se às características do esquema de distribuição, já que cada RM armazena na base de dados apenas as suas informações locais, mas necessita obter as informações de toda a topologia. Quando o RM precisa realizar uma consulta, tanto à lista de diretório quanto à de topologia, estas não são acessadas diretamente na base de dados, o que geraria um *overhead* significativo. O TS e o DS são responsáveis apenas pela atualização da base local (*cache*) do RM. Desta forma, temos uma base de dados logicamente centralizada (mas distribuída fisicamente) e replicada em cada RM.

O DS realiza dois mapeamentos associados aos nós finais (servidores físicos e VMs) e aos switches de borda: 1) armazenamento da tupla IP, MAC e porta; 2) armazenamento da tupla IP e ToR. O primeiro mapeamento é utilizado pelo RM do ToR (destino) para entregar o pacote ao servidor, realizando a associação entre o <IP, MAC> e <MAC, porta>. O segundo mapeamento é utilizado pelo RM do ToR (origem) para localizar o ToR responsável pelo IP_{dst}. No caso de máquinas virtuais endereçadas com IPs privados, adiciona-se o identificador da VLAN ou da aplicação responsável pelas VMs para garantir um mapeamento único das instâncias virtuais nos servidores físicos. O TS é responsável pela manipulação de quatro dicionários, onde três armazenam as informações da topologia, um para cada nível na árvore *fat-tree*. O quarto contém as identificações internas dos switches utilizadas pelo protocolo de descoberta para criar os *links* entre os vizinhos.

3.3. NoSQL Keyspace

Optou-se pela implementação do sistema distribuído de armazenamento do par <chave,valor> baseado no Keyspace [Trencseni and Gazso]. Além de ser uma implementação em código aberto e em Python, o que facilita a integração com o NOX, o Keyspace oferece consistência, tolerância a falhas e alta disponibilidade. O algoritmo distribuído Paxos é utilizado para manter a consistência dos dados (chave/valores) replicados, onde um quorum de nós verifica a consistência dos dados. Desta forma, pelo menos dois nós devem estar ativos. No ambiente do *cloud data center*, o número de nós do Keyspace pode ser tão grande quanto necessário, envolvendo componentes da própria infraestrutura como, por exemplo, servidores ou, mesmo, os *Rack Managers*. No ambiente de teste foram utilizados 4 nós e, na implementação, utilizou-se uma API em Python integrada aos componentes do NOX (TS e DS). Exemplos destacáveis de sistemas NoSQL [NoSQL] usados comumente nos *clusters* dos *cloud data centers* incluem o Hadoop Distributed File System (usado pelo Yahoo!), o Dynamite (baseado no Dynamo do Amazon), o Cassandra (usado pelo Facebook) e os populares MongoDB e CouchDB.

3.4. Ambiente de teste - Testbed

O ambiente de teste utilizado para validação da arquitetura proposta neste artigo é composto por 4 nós físicos em uma LAN Ethernet, cada um hospedando dois controladores

NOX (com o RM, TS e DS) e um nó hospedando o banco de dados Keyspace. Os switches e servidores são VMs, sendo 5 instâncias de switches OF e 4 nós finais Debian 4.0, configurando um total de 9 VMs em cada nó físico. A Figura 5 mostra o *testbed*, onde as linhas sólidas representam ligações diretas entre as máquinas virtuais e as linhas tracejadas representam as conexões entre as máquinas virtuais de diferentes máquinas físicas.

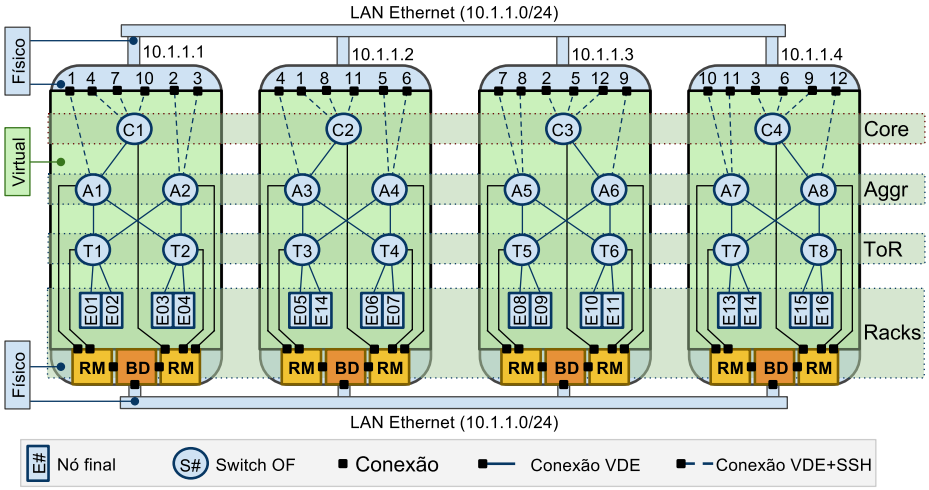


Figure 5. Ambiente de teste.

A topologia em cada máquina física é configurada com o OpenFlowVMS [Pfaff and Casado 2008], o qual dispõe de um conjunto útil de *scripts* para automatizar a criação de máquinas virtuais em rede utilizando o QEMU. *Scripts* adicionais foram desenvolvidos para distribuir o ambiente em diferentes máquinas físicas usando switches virtuais desprovidos de inteligência e baseados no VDE (*Virtual Distributed Ethernet* [Goldweber and Davoli 2008]) e conexões SSH (Secure Shell). O VDE é um *software* que emula interfaces Ethernet, como comutadores e linhas de comunicação, muito utilizado no ensino, administração e pesquisas em rede de computadores. Não se limita a ambientes virtuais e pode ser utilizado em uma ou várias máquinas distribuídas em uma rede local ou até mesmo na Internet. Já o SSH disponibiliza ao VDE um canal seguro permitindo a comunicação entre máquinas virtuais em diferentes máquinas físicas. As conexões entre máquinas físicas foram configuradas com relação de confiança, utilizando chaves assimétricas. Este mecanismo é necessário para evitar a solicitação de senhas toda vez que uma conexão SSH entre dois VDEs é realizada. O conjunto de *scripts* desenvolvido neste trabalho permite definir, rapidamente, uma topologia e automatizar a iniciação dos nós virtuais e do OF switch, incluindo a configuração de IP nos servidores, a criação de *data path* nos switches OF, a iniciação do módulo do *OpenFlow Protocol* e a conexão ao controlador.

4. Tolerância a falhas e validação experimental

Qualquer sistema de computação está sujeito a falhas, nos *cloud data center*, com milhares de switches e servidores, isto é uma situação previsível. Nesta seção analisamos e validamos experimentalmente, aqueles pontos da arquitetura proposta que podem comprometer o funcionamento do *data center*. Assim como em [Rothenberg et al. 2010], neste trabalho também não avaliamos a arquitetura quanto ao desempenho e escalabilidade, pelas limitações do ambiente virtual.

A Tabela 1 mostra uma análise dos testes realizados (avaliação prática) sobre o comportamento dos fluxos nos ToR, Aggr e Core. Verificamos o estado dos fluxos caso um switch, controlador ou nó do Keyspace venha falhar. Na sequência é apresentada uma discussão sobre os possíveis pontos de falha da arquitetura e alguns resultados da validação prática do *testbed*.

Table 1. Tolerância a falhas quanto a queda de um elemento de rede.

	Fluxo atual			Fluxo novo		
	Controlador	Switch	Nós BD[1]	Controlador	Switch	Nó BD[1]
ToR	Mantido	Interrompido	Mantido	Impossível[2]	Impossível	Criado
Aggr	Mantido	Interrompido	Mantido	Criado[3]	Criado[3]	Criado
Core	Mantido	Interrompido	Mantido	Criado[3]	Criado[3]	Criado

[1] BD Keyspace: Podem cair nós na base de dados até que fique um mínimo de 2 nós.
[2] Os pacotes que chegam, são entregues ao nó final.
[3] Criado: Novo fluxo é criado utilizando outra rota.

Rack Manager: Se o *Rack Manager* cair, o atendimento de novos fluxos saindo dos servidores dentro do *rack* fica comprometido, pois o RM é responsável por inserir as entradas no ToR que determinam o encaminhamento baseado nos iBFs. Na implementação discutida no artigo, o RM é totalmente distribuído e opera de forma independente. Desta forma, o escopo afetado é reduzido para um único *rack*. Os fluxos correntes não são interrompidos com a queda do RM e para os novos fluxos que têm como destino os servidores agregados ao ToR cujo RM falhou, os pacotes são entregues, mas as respostas não são encaminhadas, já que não existe um controlador para inserir as novas regras.

Após um evento de falha no RM, este pode reiniciar-se em questão de segundos, restabelecer a conexão com o(s) switch(es) OpenFlow, recuperar o estado atual da rede acessando os serviços TS e DS e continuar atendendo as requisições de novos fluxo de tráfego. No ambiente de *testbed* com RTT (*round-trip delay time*) médio de 19 ms (28 ms com percentil 95 após 1000 medições entre combinações de pares de servidores), o tempo médio de atraso adicional do primeiro pacote de um fluxo é de 90 ms (132 ms com percentil 95) devido ao *overhead* de ser redirecionado ao RM até que este retorne a decisão de instalação do fluxo com o mapeamento do iBF. Embora estes tempos não sejam significativos pelas limitações de um *testbed* virtualizado, vale como referência para quantificar como tolerável o *overhead* relativo introduzido pela proposta de controle de rede logicamente centralizado mas fisicamente distribuído.

Topology Service e Directory Service: Os TS e DS também são distribuídos e independentes para garantir o isolamento quando da ocorrência de uma falha. O DS possui um escopo de atuação semelhante ao RM, já que atua nos ToRs. O TS possui um escopo mais amplo, pois atua nos três níveis da topologia *fat-tree*. Por isso, o fator tempo em que o TS fica fora de operação deve ser levado em consideração. Nas falhas transitórias, nada ocorrerá com os fluxos novos e correntes, mas quando as ligações entre switches, mantidas pelo *Tree Discovery* (Seção 3.2), sofrerem *timeout*, as entradas dos filtros de Bloom dos vizinhos (Aggr e Core) serão removidas. Na implementação atual, o *timeout* é disparado após o switch não responder a três eventos consecutivos de encaminhamento de mensagens LLDP. Atualmente os pacotes LLDPs são enviados em intervalos de 10 ms.

Base de dados NoSQL: O sistema de base de dados distribuído apresenta um menor impacto, do ponto de vista da ocorrência de falhas, relativamente ao funcionamento da infraestrutura. Deve ainda ser ressaltado que o próprio sistema garante confiabilidade nas informação. A maior influência no caso da ocorrência de falhas refere-se à questão do acesso à base de dados, que pode ser feita de forma segura (*safe*) ou suja (*dirty*). Métodos *dirty* não oferecem garantias de consistência ao retornar, ou gravar, os valores, mas são mais rápidos. Estes métodos são disponibilizados pelos nós não-mestre (*no-master*) do Keyspace. Os métodos *safes* são disponibilizados apenas pelos mestres, que são mais confiáveis e possuem menos falhas. Quando um *master* cai, um nó não-mestre assume o seu lugar. Desta forma, as falhas na base de dados são transparentes para o TS e DS.

Switches OpenFlow: Quando um switch falhar, seja por um intervalo transitório ou longo, é inevitável que os fluxos correntes passando por ele sejam afetados. Com todos os caminhos habilitados e o uso do VLB para balanceamento de carga, diminuem as chances de *hotspots* e o impacto da queda de qualquer switch intermediário, já que a carga está balanceada entre todos os switches disponíveis.

Com o encaminhamento baseado nos iBFs inserido nos ToRs conforme as regras de fluxo do OpenFlow, incorporou-se um mecanismo de recuperação de falhas dos switches intermediários (Aggr e Core) semelhante ao IP *fast re-route* do MPLS [Shand and Bryant 2010]. A ideia é, com a instalação de cada novo fluxo, também instalar um segundo com prioridade menor, com um iBF alternativo que descreva uma outra rota aleatória, mas paralela, ao primeiro iBF randomicamente selecionado. Desta forma, quando ocorrer um evento de queda de switch intermediário que afete o caminho principal, o RM deve eliminar no ToR as entradas de prioridade alta afetadas pela falha. Desta forma, as alternativas começariam a funcionar com o tráfego redirecionado de forma transparente por uma combinação de switches Aggr e Core. Este mecanismo também pode ser explorado no caso da deteção de eventos de congestionamento na rota atual e não necessariamente na falha total de um switch intermediário.

Se a conectividade com controlador OpenFlow cair, após um determinado intervalo de tempo os switches podem ser configurados para descartar todos os pacotes ou entrar em modo *learning* (por defeito). Este comportamento padrão prejudica o protocolo de descoberta e não é recomendado na arquitetura proposta pois os ARPs inundariam a malha de switches e desabilitariam o protocolo de descoberta, ocasionando *loop* de pacotes LLDP. Outra alternativa seria utilizar o estado de emergência, onde regras são instaladas e utilizadas apenas em caso de falha do controlador. Uma solução ainda não especificada na versão atual do OpenFlow, consiste na utilização de controladores *slaves* (escravos). Esta e outras questões encontram-se em discussão na comunidade OpenFlow.

5. Trabalhos relacionados

O VL2 [Greenberg et al. 2009] é uma proposta de arquitetura de *data center* que oferece uma camada 2 virtual escalável. Implementa um *Directory System* (DS) que provê três funções: consultas, atualizações e mapeamentos. Apoiado nas diferentes exigências de desempenho, O DS do VL2 possui duas camadas de servidores replicados, uma otimizada para leitura e mapeamento (*directory servers*) que deve oferecer baixa latência e alta disponibilidade, e outra otimizada para escrita (*replicated state machine*), que utiliza o algoritmo de consenso Paxos para garantir a consistência das informações. Essa otimiza-

ção, onde há mais consulta do que escrita, é realizada no DS da arquitetura proposta através dos métodos *dirty* (para leitura) e *safe* (para escrita) disponíveis no Keyspace.

O Portland [Mysore et al. 2009] propõe um encaminhamento baseado na posição de pseudo endereço MAC (PMAC) atribuído aos nós finais para oferecer escalabilidade sem alterar o endereçamento Ethernet. Para isso, desenvolveu-se um protocolo de descoberta de posição dos switches na topologia (LDP) e um serviço logicamente centralizado de gerenciamento da infraestrutura de rede (*Fabric Manager* - FM). O FM mantém um estado da topologia da rede e um mapeamento entre o endereço MAC real do nó e um PMAC. Tanto o LDP quanto o FM possuem características semelhantes à arquitetura apresentada neste artigo como, por exemplo, o conhecimento prévio das posições dos switches para o encaminhamento e a manutenção de um estado global da rede.

6. Considerações finais e trabalhos futuros

No cenário tecnológico de *data centers* provendo serviços em nuvem e motivados pelos objetivos de controle (otimização das funcionalidades, customização dos serviços e aplicações, rápido processo de inovação) e baixo custo, aparece com força uma tendência de redes baseadas em equipamentos aderentes a interfaces de controle padronizadas. Porém, até esse cenário se tornar uma realidade operacional, vários desafios terão que ser contornados, entre eles, a provisão de soluções escaláveis e tolerantes a falhas, dois aspectos especialmente críticos para um serviço de rede como é o de encaminhamento de pacotes. Com esse objetivo, o presente artigo estende a proposta de arquitetura de *data centers* baseada na codificação de rotas na origem por meio de filtros de Bloom nos pacotes através da introdução de dois serviços escaláveis e tolerantes a falhas, os serviços de Diretório e Topologia, que se juntam a uma distribuição física dos elementos de controle da rede (*Rack Managers*). A distribuição do estado global da topologia e diretório torna o ambiente altamente escalável e tolerante a falhas, dois fatos corroborados pelas avaliações analíticas e experimentais apresentadas neste artigo. Em trabalhos futuros, algumas questões ligadas à tolerância a falhas deverão ser explorados com maior detalhe como, por exemplo, o tratamento eficiente dos *timeouts* nas ligações entre switches vizinhos quando da ocorrência de múltiplas falhas. Partindo da opção adotada neste trabalho por uma arquitetura de controle intermediária entre as opções totalmente centralizada e totalmente distribuída, os objetivos futuros do presente trabalho têm como foco o suporte eficiente à migração de máquinas virtuais. Esta migração eficiente é um requisito fundamental para o funcionamento otimizado dos futuros *cloud data centers*. Por último, é importante frisar que a arquitetura procurará incorporar as futuras extensões que serão agregadas à especificação do padrão OpenFlow.

Referências

Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. *SIGCOMM CCR*, 38(4):63–74.

Barroso, L. A. and Hölzle, U. (2009). *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, San Rafael, CA, USA.

Dean, J. and Ghemawat, S. (2010). MapReduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77.

Goldweber, M. and Davoli, R. (2008). VDE: an emulation environment for supporting computer networking courses. In *ITiCSE '08*, pages 138–142, New York, NY, USA.

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2009). VL2: a scalable and flexible data center network. In *SIGCOMM '09*, pages 51–62, New York, NY, USA. ACM.

Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A clean slate 4D approach to network control and management. *SIGCOMM CCR*, 35(5):41–54.

Greene, K. (2009). Software-Defined Networking. *MIT technology review*, 112(2):54.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: towards an operating system for networks. *SIGCOMM CCR*.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74.

Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. (2009). PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09*, pages 39–50, New York, NY, USA.

NoSQL. NoSQL: Your Ultimate Guide to the Non - Relational Universe! http://nosql-databases.org/.

Pfaff, B. and Casado, M. (2008). OpenFlowVMS: OpenFlow Virtual Machine Simulation. http://www.openflowswitch.org. Acesso em 25/02/2010.

Rothenberg, C. E., Macapuna, C. A. B., Verdi, F., Magalhães, M., and Zahemszky, A. (2010). Data center networking with in-packet Bloom filters. In *SBRC 2010*.

Shand, M. and Bryant, S. (2010). IP Fast Reroute Framework. RFC 5714 (Informational).

Trencseni, M. and Gazso, A. Keyspace: A Consistently Replicated, Highly-Available Key-Value Store. Whitepaper. http://scalien.com/whitepapers.

Verdi, F. L., Rothenberg, C. E., Pasquini, R., and Magalhães, M. (2010). Novas arquiteturas de data center para cloud computing. In *SBRC 2010 - Minicursos*.

Wang, F., Qiu, J., Yang, J., Dong, B., Li, X., and Li, Y. (2009). Hadoop high availability through metadata replication. In *CloudDB '09*, pages 37–44, New York, NY, USA.

Yuan, X., Nienaber, W., Duan, Z., and Melhem, R. (2007). Oblivious routing for fat-tree based system area networks with uncertain traffic demands. In *SIGMETRICS '07*, pages 337–348, New York, NY, USA. ACM.

VIII Workshop em Clouds, Grids e Aplicações

♦

Sessão Técnica 4

Segurança

Um Mecanismo Eficiente de Confiança para a Detecção e Punição de Usuários Maliciosos em Grades Peer-to-peer

Igor A. Chaves [1], Reinaldo B. Braga [1], Rossana M. C. Andrade [1], José N. de Souza [1] e Bruno Schulze [2] *

[1]Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat)
Universidade Federal do Ceará (UFC)

[2]Laboratório Nacional de Computação Científica (LNCC)

{igor,reinaldo,rossana,neuman}@great.ufc.br,schulze@lncc.br

Abstract. *Peer-to-Peer grid environments demand the correct execution of tasks to guarantee good performance. However, in these environments, there are malicious users that affect the grid performance. These users modify the tasks results and even cheat security mechanisms. Thus, it is necessary an effective mechanism able to detect and punish these malicious users. This paper presents then a reactive trust-based security mechanism that aims at detecting and punishing malicious users who corrupt the tasks results of a P2P grid. These results are obtained with simulations of grid P2P environments. We also present the results analysis that shows the proposed mechanism is efficient and manages to detect and punish 100 % of the stations that modify the tasks results of the P2P grid.*

Resumo. *A correta execução das tarefas em ambientes de grades peer-to-peer (P2P) é fundamental para o seu bom desempenho. Entretanto, nestes ambientes, existem usuários maliciosos que prejudicam o desempenho da grade modificando os resultados das tarefas e, até mesmo, burlando os mecanismos de segurança. Desta forma, faz-se necessário um mecanismo eficiente capaz de detectar e punir os usuários maliciosos da grade P2P. Este trabalho apresenta um mecanismo reativo de segurança baseado em confiança, tendo como objetivo principal detectar e punir os usuários maliciosos que corrompem o resultado das tarefas da grade P2P. Os resultados apresentados foram obtidos através de simulações de ambientes de grades P2P. Ao analisar os resultados, é possível concluir que o mecanismo proposto é eficiente, detectando e punindo até 100% as estações que modificam os resultados das tarefas da grade P2P.*

1. Introdução

As grades computacionais são formadas a partir de recursos computacionais heterogêneos e distribuídos geograficamente, que possibilitam a criação de um ambiente com alto poder de processamento e de armazenamento [Foster e Kesselman, 2004]. O ambiente de grades fornece recursos para diversas aplicações, tais como armazenamento, análise e virtualização de dados. As grades *peer-to-peer* (P2P) se caracterizam por sua infraestrutura descentralizada, o que aumenta a escalabilidade quando comparada com as grades convencionais [Marsh et al., 2008]. Elas também se caracterizam pelo fato de

*Este trabalho foi realizado com recursos do CNPq (projeto SIMEGRID)

os usuários, além de doarem seus recursos, também poderem executar suas aplicações na grade P2P [Uppuluri et al., 2005]. Visto que a grade P2P pode ser composta por máquinas heterogêneas, localizadas em diferentes domínios de redes e com variados tipos de usuários, é possível, então, que existam usuários maliciosos, cujo comportamento pode prejudicar o desempenho da grade. Um dos exemplos mais conhecidos é o caso do SETI@home [Seti@Home, 2009], no qual voluntários alteravam sua quantidade de trabalhos realizados com o objetivo de aumentar sua reputação no *ranking* de maiores colaboradores do projeto SETI@home. Outro exemplo bastante conhecido é o dos usuários maliciosos que modificam os resultados das tarefas da grade. Estes usuários têm como objetivo principal enviar uma resposta aleatória para as estações da grade, pois isso faz com que a estação maliciosa não gaste os seus recursos processando uma tarefa e não seja excluído da grade, por estar respondendo-a. Já outros visam apenas prejudicar o funcionamento da grade.

Diversas soluções já foram propostas com o objetivo de aumentar a precisão dos resultados das tarefas executadas pela grade. Um dos exemplos é a votação majoritária, na qual uma tarefa é replicada e enviada para vários usuários executarem e, de acordo com as respostas recebidas, o usuário que enviou a tarefa decide se o resultado desta é aceito ou não [Sarmenta, 2001]. Outra solução utilizada é o *testjob*, que corresponde à submissão de uma tarefa de resultado já conhecido para um usuário executar. A resposta desse usuário é comparada com o resultado conhecido e, a partir daí, é possível saber se ele está executando de forma correta as tarefas enviadas para ele.

Baseados nessas primeiras soluções, diversos mecanismos foram propostos para tentar identificar usuários maliciosos, como [Sarmenta, 2001], que combina *testjobs* com votação majoritária com o objetivo de alcançar uma probabilidade mínima de o resultado da tarefa estar correto. Outro modelo é proposto em [Martins et al., 2006], no qual usuários maliciosos são detectados a partir de um mecanismo baseado em *testjobs* e reputação. Os usuários possuem três níveis de confiabilidade: executores, testadores e ultra-confiáveis (UR). Através do envio de *testjobs* entre esses usuários é possível identificar usuários maliciosos. A solução proposta em [Zhao e GauthierDickey, 2005] sugere a criação de pacotes, chamados de *Quiz*, contendo diversas tarefas que serão enviadas para execução. Essas tarefas são divididas em duas: tarefas normais e *testjobs*, com o objetivo de dificultar a identificação de *testjobs* pelos usuários maliciosos. A partir do resultado dos *testjobs* que vão dentro do pacote, o usuário decide aceitar ou não os resultados das aplicações restantes do pacote.

Além dos mecanismos de detecção e punição dos usuários maliciosos, existem propostas baseadas em confiança para detectar os usuários cujo comportamento prejudica o desempenho da grade [Azzedin e Maheswaran, 2002] [Virendra et al., 2005] [Yu et al., 2004] [Liu e Issarny, 2004a] [Liu e Issarny, 2004b]. Esses mecanismos tratam de diferentes formas as informações adquiridas localmente e as informações obtidas por meio dos outros usuários da grade.

Visando evitar a presença de usuários maliciosos na grade P2P, neste trabalho é proposto um mecanismo de confiança para a detecção e a punição de usuários maliciosos. A proposta apresenta uma solução de segurança reativa baseada nos resultados de *testjobs* inseridos em tarefas normais e analisados em janelas independentes de tempo. A solução foi implementada e analisada em um simulador de grades computacionais. A partir dos

resultados obtidos, pode-se concluir que a proposta é eficiente para detectar e punir os usuários que modificam as respostas das tarefas.

Este trabalho está organizado da seguinte forma: na Seção 2 são apresentados e discutidos os trabalhos relacionados; na Seção 3 é apresentado um mecanismo de segurança baseado em confiança, como forma de solucionar os problemas apresentados; na Seção 4 são definidas as variáveis do ambiente de simulação e são mostrados e discutidos os resultados dessas simulações; finalmente, na Seção 5 são apresentadas as conclusões e os trabalhos futuros.

2. Trabalhos Relacionados

Sarmenta propõe um modelo baseado em votação majoritária e em *testjob* [Sarmenta, 2001]. Nesta proposta, é necessário que uma quantidade mínima de respostas corretas seja atingida para que o resultado da tarefa seja considerado correto e, portanto, aceito pela estação de origem. Entretanto, a utilização do mecanismo de votação majoritária pode gerar um alto *overhead*, o que diminui o desempenho da grade computacional.

Com base na solução de *testjobs*, [Martins et al., 2006] propõe um mecanismo hierárquico. Para detectar um usuário malicioso, os usuários testadores analisam os usuários executores e repassam as informações de detecção para os usuários ultra-confiáveis (UR). Desta forma, os usuários UR decidem por detectar/punir ou não o usuário em questão a partir das informações passadas pelos usuários testadores. Para aumentar a segurança nas informações passadas, os usuários testadores também são analisados através de *testjobs* enviados pelos usuários UR. Além disso, os usuários UR são analisados por outros usuários UR. No entanto, mecanismos de detecção de usuários maliciosos que utilizam informações de terceiros são passíveis de detecções/punições incorretas, pois usuários maliciosos podem enviar falsas acusações sobre os usuários normais da grade.

Ao observarem esse tipo de usuário malicioso, que difama os usuários normais da grade, diversas propostas de detecção de intrusão adicionaram modelos de confiança em seus mecanismos de detecção e punição de usuários maliciosos [Azzedin e Maheswaran, 2002] [Virendra et al., 2005] [Yu et al., 2004] [Liu e Issarny, 2004a] [Liu e Issarny, 2004b]. Nestes trabalhos, são utilizadas formulações matemáticas que representam os modelos de confiança. Para relacionar a detecção e a punição com os modelos de confiança são utilizadas as informações locais de detecção, assim como, as informações de confiança passadas por terceiros. Desse modo, o mecanismo pode decidir por punir ou não um determinado usuário. Como dito anteriormente, quando informações de confiança são recebidas de terceiros, existe a possibilidade dos usuários normais serem detectados como maliciosos e de usuários maliciosos não serem detectados.

Com o objetivo de solucionar os problemas citados anteriormente, neste artigo é proposto um mecanismo de segurança baseado em confiança para detectar e punir usuários maliciosos. Esse mecanismo utiliza *testjobs* camuflados nas tarefas normais da grade. Estes *testjobs* são analisados em diferentes intervalos de tempo. Para aumentar a eficiência da proposta, as informações de confiança passadas pelos usuários da grade P2P não são utilizadas diretamente para detectar os usuários maliciosos. Estas informações são usadas para definir o tamanho do intervalo de tempo da análise de *testjobs*. Desta forma, pode-se definir um grau de tolerância para os testes que estão sendo realizados na

grade, evitando a ocorrência de falsas detecções, ou seja, de falso-positivos.

3. Mecanismo Proposto

Neste trabalho é apresentada uma solução de segurança reativa baseada nos resultados de *testjobs* enviados para os usuários em função do tempo. Como o principal objetivo da proposta é identificar e punir usuários de forma reativa, é considerada a existência de um modelo de segurança preventivo de autenticação. As verificações de comportamento são realizadas de forma independente, desse modo, cada usuário pode avaliar o comportamento dos outros usuários que estão fornecendo seus recursos para a grade. A partir do mecanismo de segurança proposto é possível detectar e punir os usuários maliciosos cujo comportamento inadequado compromete o desempenho da grade P2P.

3.1. Cálculo da Confiança

Diversos modelos de confiança existentes na literatura [Azzedin e Maheswaran, 2002] [Virendra et al., 2005] [Yu et al., 2004] [Liu e Issarny, 2004a] [Liu e Issarny, 2004b] utilizam as informações passadas pelos usuários sobre o comportamento de terceiros. Além disso, consideram as informações locais de detecção para inserir alguma punição aos usuários maliciosos. Porém, esses modelos recaem na mesma fragilidade: a possível geração de falso-positivos. Isso ocorre porque o modelo de segurança utiliza as informações passadas pelos outros usuários da grade para a detecção de usuários maliciosos. Entretanto, as informações de confiança passada pelos usuários da grade podem não ser verdadeiras.

Portanto, partindo desses argumentos, é proposta uma métrica de confiança, representada por C. Essa métrica é utilizada para detectar e punir os usuários maliciosos e é calculada somente a partir de testes realizados pelo próprio usuário. Assim, evita-se a punição dos usuários normais devido às falsas informações enviadas por usuários difamadores.

Os testes utilizados na proposta seguem o modelo de *Quiz* [Zhao e GauthierDickey, 2005]. O *Quiz* é baseado na criação de pacotes de tarefas, que são enviados para serem executados na grade. As tarefas que irão compor esse pacote são de dois tipos: tarefas normais e *testjobs*. Esse modelo tem como objetivo dificultar a identificação dos *testjobs* pelos usuários maliciosos, já que são enviados diferentes testes para os usuários.

Para o cálculo da confiança C, apresentado nas Equações 1 e 2, são atribuídos pesos a cada um dos resultados dos testes. Esses pesos formam uma progressão aritmética crescente de razão igual a 1, sendo r_1 referente ao resultado do primeiro teste efetuado e o r_k referente ao resultado do *k-ésimo* teste efetuado. Portanto, se o resultado do teste é correto, então $r = 1$.

$$C = \frac{1 \cdot r_1 + 2 \cdot r_2 + \ldots + (n-1) \cdot r_{n-1} + n \cdot r_n}{1 + 2 + ... + (n-1) + n} \tag{1}$$

$$C = \frac{2[1 \cdot r_1 + 2 \cdot r_2 + \ldots + (n-1) \cdot r_{n-1} + n \cdot r_n]}{n \cdot (n+1)} \tag{2}$$

Um peso maior aos resultados dos testes mais recentes, ou seja, os últimos testes realizados têm uma maior contribuição no cálculo da confiança e, consequentemente, na

detecção e punição de usuários maliciosos. A distribuição de pesos se torna mais clara quando o número de testes realizado é grande e os pesos referentes aos resultados dos testes são iguais a 1. Este é o caso, por exemplo, de um usuário que realiza 30 tarefas normalmente e que, a partir de um determinado instante, decide agir maliciosamente modificando o resultado das tarefas processadas por ele. Pode-se observar que esse tipo de usuário está tentando burlar o mecanismo de segurança. Com base nesse exemplo, é observado que, se os resultados dos testes obtidos mais recentemente têm o mesmo peso dos resultados obtidos inicialmente, então, ocorrerá uma demora na detecção e punição deste usuário malicioso.

Ao observar este exemplo, é possível perceber a importância em utilizar a ponderação com peso maior nos eventos detectados recentemente. Desta forma, a partir do momento em que algum usuário começa a modificar o resultado das tarefas, esses resultados terão um peso maior no cálculo da confiança em relação aos resultados obtidos anteriormente, facilitando a detecção e a punição de usuários que modificam o resultado das tarefas da grade.

3.2. Intervalo de Tempo

Ao observar a forma como é calculada a confiança, nota-se que, na medida em que o tempo passa, o número de tarefas executadas e o número de testes realizados aumentam. Assim, n aumenta e a contribuição de cada teste no cálculo da confiança diminui, fazendo com que o mecanismo se torne mais tolerante aos erros. Isso significa que, quando n é muito grande, existe uma tendência de o mecanismo depender de um maior tempo de interação para detectar e punir os usuários maliciosos que modificam o resultado das tarefas.

Partindo dessa análise, observa-se a necessidade de ajustar a quantidade de testes a serem utilizados para o cálculo da confiança. Propõe-se, então, a utilização de um intervalo de tempo para delimitar um conjunto de testes a serem considerados para o cálculo da confiança. Desse modo, os testes realizados anteriormente a esse determinado intervalo de tempo são desconsiderados para o cálculo da confiança e, consequentemente, para uma eventual detecção.

Analisando a variação desse intervalo de tempo pode-se fazer algumas considerações. A primeira delas é que quanto maior for o valor do Δt, maior será a quantidade de testes utilizados para o cálculo da confiança. Consequentemente, maior será a tolerância a erros, tornando a proposta favorável somente aos usuários que são considerados confiáveis pela grade. Contrário a essa situação, temos que quanto menor for o valor do Δt, menor será a quantidade de testes considerados para o cálculo da confiança. Isso acarreta em uma menor tolerância aos erros, o que é bem interessante para usuários de comportamento malicioso.

Após essa análise pode-se concluir que o principal desafio está relacionado ao cálculo desse intervalo de tempo. Para tentar resolver esse desafio, propõe-se que Δt seja calculado a partir da multiplicação da função $f(C_{grade})$ por Δt_{ant}, como mostra a Equação 3. Sendo C_{grade} calculado a partir da confiança que os usuários da grade passam sobre o usuário a ser avaliado e Δt_{ant} o intervalo de tempo imediatamente anterior a esse.

$$\Delta t = f(C_{grade}) \cdot \Delta t_{ant} \tag{3}$$

3.2.1. Cálculo da função *f*

A função f funcionará como um fator multiplicativo que será responsável pelo aumento ou diminuição do intervalo de tempo de um determinado usuário. Esse fator será calculado utilizando informações de confiança, C_{grade}, passadas pelos usuários da grade.

O objetivo da função f é variar Δt de modo a obtermos uma melhor capacidade de análise de um determinado usuário. No caso de $f > 1$, haverá um aumento em Δt, o que é interessante, como discutido anteriormente, para usuários considerados confiáveis pela grade. No caso de $f < 1$, haverá uma diminuição do Δt, o que interessa aos usuários de comportamento malicioso. Para definir se um usuário é considerado confiável pela grade definimos um limiar L, onde se $C_{grade} < L$ o usuário é considerado malicioso, e se $C_{grade} > L$ o usuário é considerado confiável pela grade.

A partir dessa discussão é feito um esboço do gráfico de f apresentado na Figura 1. Nota-se que se $C_{grade} > L \Rightarrow f > 1$, se $C_{grade} < L \Rightarrow f < 1$ e se $C_{grade} = L \Rightarrow f = 1$. Sendo C_{grade} a confiança da grade em um usuário e L o limite que define a confiabilidade de um usuário, percebe-se que quando $C_{grade} > L$, ou seja, quando o usuário é considerado normal pela grade, tem-se um aumento no valor de Δt. Quando $C_{grade} < L$, ou seja, quando o usuário é considerado malicioso pela grade, tem-se uma redução no valor de Δt. Por fim, quando $C_{grade} = L$, o valor do Δt não é alterado. Nota-se, também, que f é limitada superiormente por f_{max}, já que $C_{grade} \leq 1$. Esse valor máximo ocorre quando a confiança da grade em certo usuário for máxima, ou seja, quando $C_{grade} = 1$, e por isso o valor de Δt terá o maior aumento possível.

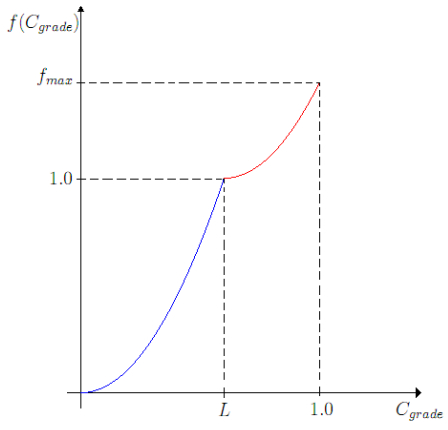


Figura 1. Esboço do gráfico da função f.

A partir do gráfico apresentado na Figura 1 pode-se observar o comportamento de f quando C_{grade} está próximo de L. Quando $C_{grade} > L$ e mais próximo de L significa que a confiança desse usuário está próxima do limiar, então, é importante que o Δt para esse usuário aumente lentamente. Portanto, se o usuário está próximo da faixa para ser considerado malicioso, não é adequado que a tolerância desse usuário seja am-

pliada rapidamente. De forma oposta, quando $C_{grade} < L$, porém não muito menor do que L, isso significa que o usuário é considerado não confiável pela grade. Apesar da confiança do usuário estar próxima do limite, isso acarreta em uma queda mais abrupta do Δt. Portanto, caso esse usuário esteja em um grande intervalo de tempo e comece a agir de forma maliciosa, é importante que ocorra uma diminuição mais acentuada do Δt. Assim, é importante que ocorra uma rápida redução da tolerância contra os erros para esse usuário, diminuindo o tempo de detecção. Essa queda acentuada do valor de f, que, por conseguinte, gera uma queda acentuada do valor de Δt.

As duas curvas observadas no gráfico da Figura 1 como parte da função f são definidas como duas parábolas. A partir dessas duas parábolas, é definida a função f, apresentada na Equação 4. Sendo a_1, b_1 e c_1 coeficientes da primeira parábola e a_2, b_2 e c_2 coeficientes da segunda parábola. Desta forma, é necessário encontrar cada um desses coeficientes para que f se comporte tal como apresentado no gráfico.

$$f(C_{grade}) = \begin{cases} a_1 \cdot C_{grade}^2 + b_1 \cdot C_{grade} + c_1, & se\ \ C_{grade} < L \\ \\ a_2 \cdot C_{grade}^2 + b_2 \cdot C_{grade} + c_2, & se\ \ C_{grade} \geq L \end{cases} \tag{4}$$

Esses coeficientes são encontrados através da resolução de um sistema de equações, construído a partir dos pontos observados nos gráfico da Figura 1, sendo f_{max} e L duas constantes a serem definidas pelo usuário. Para encontrar os coeficientes da primeira equação, são utilizados como pontos: a origem, o x do vértice e o ponto $(L, 1)$.

$$\begin{cases} f(0) = 0 & \rightarrow c_1 = 0 \\ \\ X_v = \frac{-b_1}{a_1} & \rightarrow \frac{-b_1}{a_1} = 0 \qquad \rightarrow b_1 = 0 \\ \\ f(L) = 1 & \rightarrow a_1 \cdot L^2 = 1 \quad \rightarrow a_1 = \frac{1}{L} \end{cases} \tag{5}$$

Para encontrar os coeficientes da segunda equação também são utilizados três pontos: x do vértice, y do vértice e o ponto $(1, f_{max})$.

$$\begin{cases} X_v = L = \frac{-b_2}{a_2} \\ \\ Y_v = 1 = \frac{-b_2^2 + 4a_2 \cdot c_2}{4a_2} \\ \\ f(1) = f_{max} = a_2 + b_2 + c_2 \end{cases} \Rightarrow \begin{cases} a_2 = \frac{-b_2}{L} \\ \\ b_2 = \frac{2L\ (f_{max} - 1)}{-L^2 + 2L - 1} \\ \\ c_2 = \frac{2 - L \cdot b_2}{2} \end{cases} \tag{6}$$

A partir da definição das duas constantes L e f_{max} são encontrados, através de uma simples substituição, os coeficientes a, b e c necessários para achar a função $f(C_{grade})$, tal como apresentada no gráfico da Figura 1.

4. Parâmetros e Resultados

Antes de apresentar o ambiente de simulação, é importante destacar os tipos de usuários maliciosos utilizados nesse ambiente. São utilizados dois tipos de comportamento malicioso que esses usuários poderão ter: modificar o resultado das tarefas com

uma determinada probabilidade e enviar falsas reputações de usuários para a grade. A partir disso, são definidos cinco tipos de usuários maliciosos, classificados no ambiente da seguinte forma:

- **Modificadores:** esse tipo de usuário malicioso corrompe o resultado das tarefas enviadas para ele. Para analisar as variações deste tipo de usuário, foram observadas diferentes probabilidades desses usuários se comportarem de forma maliciosa.
- **Inteligentes:** esse usuário ganha confiança da grade agindo normalmente por um período, executando normalmente as tarefas enviadas para ele. Após certo tempo, ele começa a corromper o resultado das tarefas também com certa probabilidade.
- **Difamadores:** usuários que difamam outros usuários da grade. Em outras palavras, estes usuários enviam falsas informações de confiança sobre terceiros para os outros usuários da grade.
- **Modificadores e Difamadores:** usuários que, além de modificar o resultado das tarefas com certa probabilidade, difamam os usuários da grade.
- **Inteligentes e Difamadores:** usuários maliciosos inteligentes, mas que também enviam falsas informações de confiança para a grade.

Para validar o modelo de segurança proposto, foi simulado um ambiente de grades p2p utilizando o simulador Gridsim [Buyya e Murshed, 2002]. Nessa seção são apresentados os parâmetros e os resultados do ambiente de simulação do trabalho.

4.1. Ambiente de Simulação

Para avaliar o impacto da utilização do mecanismo de segurança proposto, foram simulados diferentes cenários de grades p2p. Os cenários simulados utilizaram um total de 60 usuários. Desse total, 15 usuários eram maliciosos, ou seja, 25%. Cada usuário gera 1500 pacotes (tarefa + *testjob*), que são executados pela grade p2p. Os usuários executores das tarefas são escolhidos aleatoriamente pelos usuários que as submetem. Durante a simulação, a partir do instante em que um usuário detecta um terceiro como malicioso, ele não envia nem recebe pacotes relacionados ao usuário malicioso. Portanto, o usuário malicioso fica bloqueado para realizar qualquer tipo de execução com o usuário que o detectou. Para cada cenário foram realizadas 30 simulações e todos os resultados são apresentados com um intervalo de confiança de 95%.

A porcentagem de 25% de usuários maliciosos foi escolhida depois de observado que essa quantidade de usuários maliciosos já afeta significativamente o desempenho da grade. O gráfico da Figura 2 mostra o resultado de um cenário de grades p2p sem a utilização do mecanismo de segurança. Nesse cenário é variada a quantidade de usuários maliciosos modificadores, assim como a probabilidade desses usuários modificarem o resultado das tarefas. Desta forma, é possível apresentar o percentual de tarefas da grade com resultados alterados. Nota-se que, com 25% dos usuários maliciosos, a quantidade de tarefas com resultados corrompidos variam de $2,5\%$ a 25%, dependendo da probabilidade com que esse usuário modifica o resultado da tarefa. Isso comprova que, com essa porcentagem de usuários maliciosos, o índice de tarefas alteradas já é bem elevado e já prejudica consideravelmente o desempenho a grade.

Para a simulação, as constantes f_{max} e L foram definidas com os valores 1.5 e 0.7, respectivamente. Isso significa que o aumento máximo de Δt será de 50%. O limiar

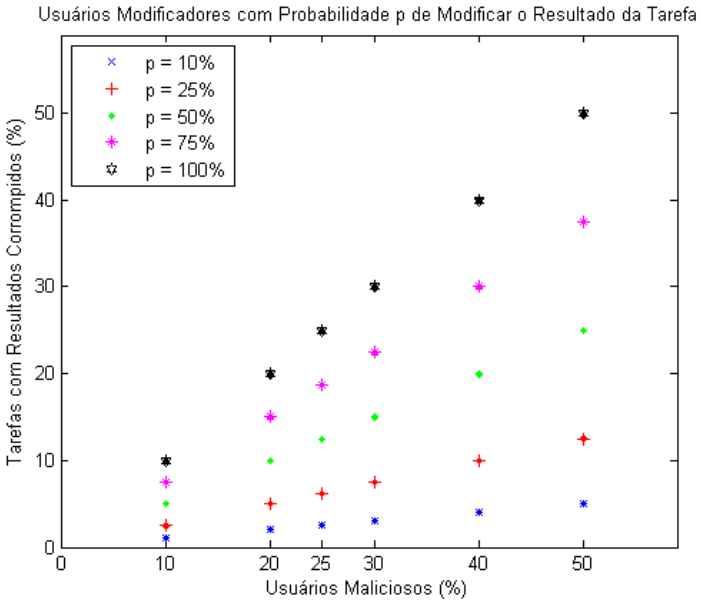


Figura 2. Percentual de tarefas corrompidas em um cenário sem mecanismos de segurança.

de confiança L escolhido deve-se ao seguinte fato: como a confiança local é baseada numa fração da quantidade de testes respondidos corretamente pela quantidade total de testes enviados, tem-se que essa confiança será, em média, a fração de testes respondidos corretamente. Dependendo dos pesos de cada um desses resultados, a confiança pode ser maior ou menor do que a fração. Ao considerar 25% das tarefas respondidas erroneamente uma porcentagem bem prejudicial para grade P2P, pode-se perceber que, se $L = 0.7$, o mecanismo ainda é tolerante. Essa tolerância é importante, pois caso aconteça algum erro, seja de *hardware* ou *software*, o usuário normal não será punido como malicioso.

Foram simulados cenários com cada um dos usuários maliciosos apresentados. Para os usuários maliciosos que modificam o resultado das tarefas, a probabilidade de eles modificarem o resultado das tarefas foi variada em 10%, 25%, 50%, 75% e 100%. Para os usuários maliciosos que difamam outros usuários, foi definido que eles escolheriam aleatoriamente 50% dos usuários normais da grade para difamar.

$$C_{grade,A} = \frac{\sum (C_{j,A} \cdot C_j)}{\sum C_j} \tag{7}$$

Para o cálculo de C_{grade} é utilizado um modelo de média ponderada apresentado em [Braga et al., 2009], pois esse foi um dos modelos que obteve melhores resultados nas análises realizadas. Nesse modelo, é obtida a média das informações de confiança passadas pelos usuários da grade, ponderada pela confiança nesse usuário que está passando essa informação. Por exemplo, para calcular o C_{grade} de um usuário A, como mostra a Equação 7, são utilizadas as informações de confiança passadas pelos usuários da grade em relação ao usuário A. Desta forma, são consideradas somente as informações passadas pelos usuários que são considerados localmente confiáveis, ou seja, quando $C_j \geq L$. Assim, nota-se que quanto maior for a confiança em um determinado usuário j, maior será a contribuição da informação passada por esse usuário no cálculo de C_{grade}.

4.2. Resultados

Os resultados são analisados a partir de gráficos que mostram a quantidade de usuários maliciosos detectados durante o tempo de simulação, utilizando o mecanismo de segurança proposto. A partir dos resultados, são discutidos os impactos que o mecanismo traz para a grade p2p.

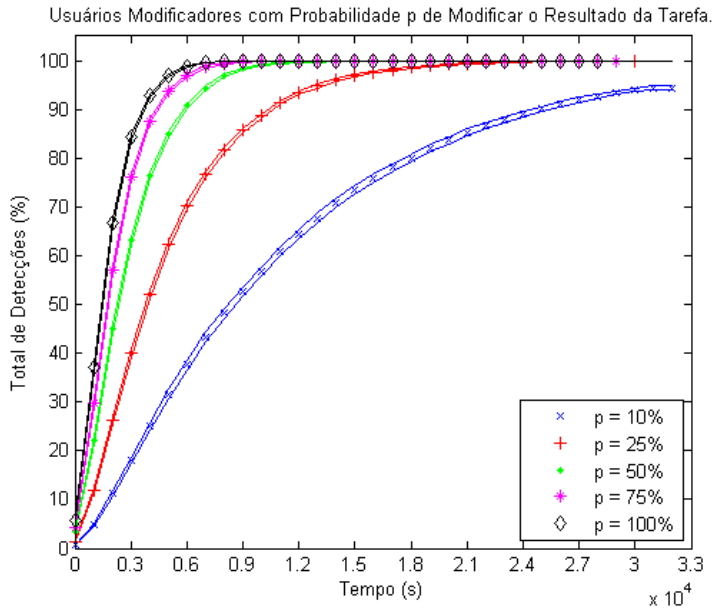


Figura 3. Total de detecções em um cenário com 25% de usuários modificadores.

Usuários Modificadores

De acordo com os resultados apresentados no gráfico da Figura 3, é possível observar que os usuários que mais prejudicam a grade são detectados mais rapidamente do que os usuários com uma menor probabilidade de corromper as tarefas. Nota-se que no instante de tempo igual a $9 \cdot 10^3$ segundos de simulação, praticamente todos os usuários com probabilidade $p = 100\%$, $p = 75\%$ e $p = 50\%$ são detectados e punidos. Já no instante de tempo igual a $18 \cdot 10^3$ segundos, são detectados quase todos os usuários com $p = 25\%$. Até mesmo os usuários com menor probabilidade de corromper o resultado das tarefas, $p = 10\%$, são detectados em até 94% dos casos, apesar de demandarem um tempo maior para que isso ocorra. Nota-se também que a detecção dos usuários com $p = 10\%$ poderia chegar a 100% se um tempo maior de simulação for considerado.

Usuários Inteligentes

No outro cenário, os usuários maliciosos executam normalmente as tarefas da grade até o instante de tempo igual a $15 \cdot 10^3$ segundos. A partir desse momento, eles passam a corromper o resultado das tarefas com uma probabilidade p. Ao observar a Figura 4 nota-se uma semelhança das curvas de detecções com os resultados da Figura 3.

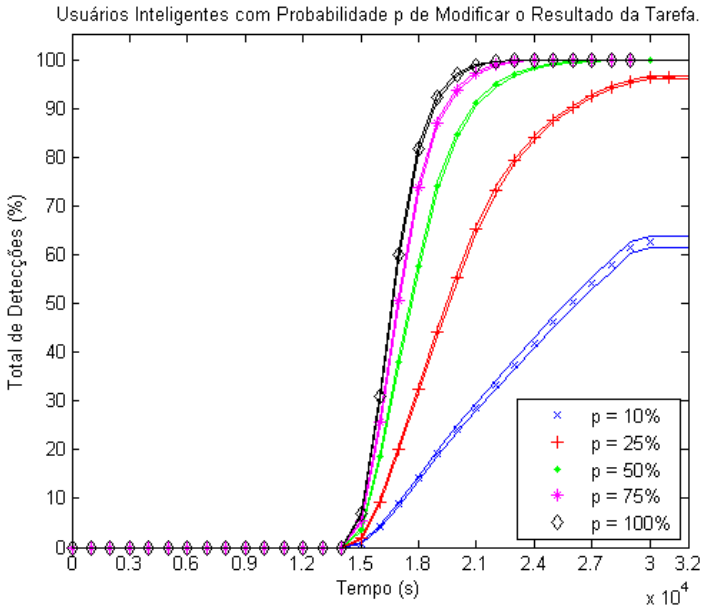


Figura 4. Total de detecções em um cenário com 25% de usuários inteligentes.

Isso significa que, apesar desse tipo de usuário tentar burlar o mecanismo de segurança, o tipo de comportamento é solucionado pelo mecanismo proposto. Isso acontece devido à utilização das janelas de tempo, sem memória, em conjunto com a ponderação utilizada para o cálculo da confiança local. Assim, mesmo que um usuário se comporte corretamente até um determinado instante, caso ele inicie algum comportamento malicioso, ele será detectado e punido.

Usuários Difamadores

Foram simulados cinco cenários diferentes com esse tipo de usuário, variando a quantidade em 10%, 20%, 30%, 40% e 50%. Em nenhum dos cenários analisados foram detectados falso-positivos. Usuários difamadores também não foram detectados. Isso ocorreu porque o mecanismo proposto não utiliza diretamente a confiança passada pelos usuários da grade para a detecção de usuários maliciosos. A confiança da grade é utilizada para calcular o intervalo de tempo no qual o usuário será analisado, definindo assim, certa tolerância ao usuário analisado. Isso significa que uma baixa confiança da grade em um usuário não causa, necessariamente, uma detecção, mas sim uma menor tolerância nesse usuário. O que denota que o mecanismo proposto é robusto para esse tipo de comportamento malicioso. Pode-se dizer, também, que o desempenho da grade não é afetado nesse cenário, pois o usuário difamador somente difama os usuários normais da grade, não alterando o resultado das tarefas executadas por ele.

Usuários Modificadores e Difamadores

Nos resultados apresentados na Figura 5, nota-se uma semelhança com os resultados da Figura 3. Apesar de os usuários maliciosos difamarem os usuários normais da grade, esse comportamento não terá um grande impacto no resultado final, pois no início da grade, o intervalo de tempo ainda é pequeno. Portanto, eles são detectados por serem

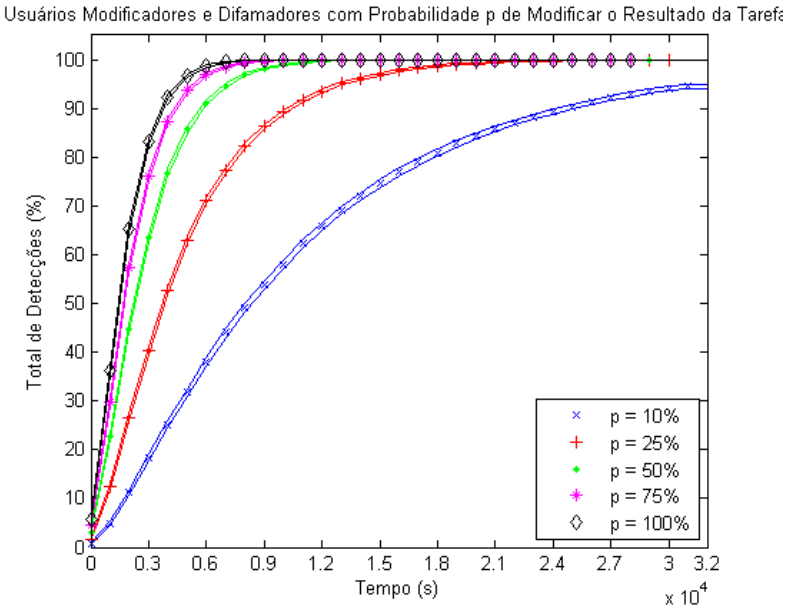


Figura 5. Total de detecções em um cenário com 25% de usuários modificadores e difamadores.

modificadores logo no início, dificultando a realização dos comportamentos de difamação por parte destes usuários.

Usuários Inteligentes e Difamadores

Os resultados desse cenário são apresentados na Figura 6. Apesar da semelhança com os resultados na Figura 4, nota-se que no cenário onde os usuários maliciosos difamam os usuários da grade, ocorre uma detecção mais rápida. Como esse tipo de usuário malicioso difama os usuários normais e, inicialmente, todos os usuários se comportam normalmente, então nota-se que o intervalo de tempo dos usuários não aumenta como no outro cenário. Consequentemente, a tolerância tende a ser menor. Por exemplo, ao observar o resultado no instante de tempo igual a $18 \cdot 10^3$ segundos, nota-se que aproximadamente 44% dos usuários maliciosos com $p = 25\%$ são detectados. No entanto, ao observar o mesmo ponto na Figura 4 nota-se que, aproximadamente, 32% dos usuários maliciosos com $p = 25\%$ são detectados. Isso ocorre porque a confiança passada pelos usuários da grade é utilizada para o cálculo do intervalo de tempo em que um usuário será analisado. Em outras palavras, o intervalo de tempo para a análise do usuário é menor, o que acarreta em uma menor tolerância aos erros e, consequentemente, em uma detecção mais rápida de usuários que corrompem o resultado das tarefas.

Ao observar os gráficos apresentados, pode-se perceber que a proposta é eficiente na detecção e punição de estações maliciosas em cenários de grades P2P. É importante destacar que esta proposta pode ser utilizada em outros cenários de grades computacionais, assim como em outros tipos de cenários que exigem uma relação de confiança entre as estações da rede, tais como as redes *ad hoc* sem fio.

5. Conclusão

Nesse artigo foi proposto e analisado um mecanismo de segurança baseado em confiança para grades P2P. Esse mecanismo tem como objetivo principal detectar e

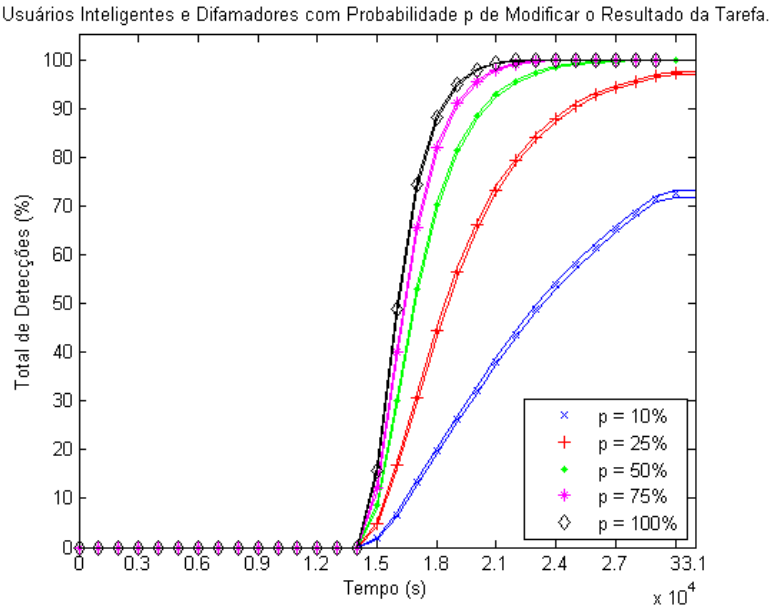


Figura 6. Total de detecções em um cenário com 25% de usuários inteligentes e difamadores.

punir usuários maliciosos que corrompem o resultado das tarefas submetidas para grade. Além disso, foi implementada uma funcionalidade que evita falsas punições na grade, provocadas pelos usuários difamadores. O mecanismo calcula a confiança a partir de informações obtidas localmente durante uma janela de tempo. Essa janela de tempo define a tolerância para cada usuário baseada nas informações de confiança passadas pelos usuários da grade.

A partir dos resultados obtidos pôde-se concluir que o mecanismo é eficiente contra os usuários que modificam o resultado das tarefas com uma determinada probabilidade, conseguindo detectar e punir 100% dos usuários com probabilidade $p \geq 25\%$ de corromper o resultado da tarefa e até 94% dos usuários com $p = 10\%$. O mecanismo se mostrou eficiente também contra os usuários maliciosos que agem normalmente por um período e, após certo instante de tempo, começam a corromper o resultado das tarefas. A partir dos resultados, pode-se perceber que são detectados até 100% desses usuários maliciosos, chamados de inteligentes. Em todos os cenários o mecanismo se mostrou robusto aos usuários maliciosos que difamam os usuários da grade, visto que em nenhum momento são gerados falso-positivos nesses cenários. Isso porque no mecanismo proposto não há a utilização direta das informações de confiança passadas pelos usuários da grade para a detecção de usuários maliciosos. Como essas informações não são diretamente utilizadas para decidir se o usuário é malicioso ou não, então não há detecção de usuários normais como maliciosos.

Apesar do conluio ser um problema existente e ainda sem solução para grupos formados por mais de 50% das estações da grade, esse artigo assume a não existência desse tipo de ataque. Portanto, as estações não agirão em conjunto para tentar enganar o mecanismo de detecção e punição da grade. Como trabalho futuro, a proposta está sendo implementada em um ambiente real de grades computacionais *peer-to-peer*, por meio dos recursos computacionais do projeto SIMEGRID [SIMEGRID, 2009]. Além disso, estão

sendo analisados outros métodos de punição para serem avaliados em conjunto com a proposta apresentada.

Referências

Azzedin, F. e Maheswaran, M. (2002). Evolving and managing trust in grid computing systems. *IEEE Canadian Conference on Electrical Computer Engeneering*.

Braga, R., Chaves, I., Andrade, R., Souza, J. e Schulze, B. (2009). Modelos probabilísticos de confiança para grades computacionais ad hoc. *Workshop on Grid Computing and Applications (WCGA)*.

Buyya, R. e Murshed, M. (2002). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Journal of Concurrency and Computation: Practice and Experience (CCPE)*.

Foster, I. e Kesselman, C. (2004). *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier, segunda edição.

Liu, J. e Issarny, V. (2004a). Enhanced reputation mechanism for mobile ad hoc networks. *Proceedings of iTrust*.

Liu, J. e Issarny, V. (2004b). A robust reputation system for p2p and mobile ad hoc networks. *Second Workshop on Economics of Peer-to-Peer Systems*.

Marsh, M., Kim, J., Nam, B., Lee, J. e Ratanasanya, S. (2008). Matchmarking and implementation issues for a p2p desktop grid. *Parallel and Distributed Processing*. IEEE International Symposium on.

Martins, F., Maia, M., Andrade, R., Santos, A. e Souza, J. (2006). Detecting malicious manipulation in grid environments. *IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*.

Sarmenta, L. (2001). Sabotage-tolerance mechanisms for volunteer computing systems. *First IEEE/ACM International Symposium on Cluster Computing and the Grid. Proceedings*.

Seti@Home (2009). http://setiathome.berkeley.edu/.

SIMEGRID (2009). Simulações médicas em grids. *http://comcidis.lncc.br/index.php/SIMEGRID*.

Uppuluri, P., Jabisetti, N., Joshi, U. e Lee, Y. (2005). P2p grid: service oriented framework for distributed resource management. *IEEE International Conf. on Services Computing*, p. 347–350.

Virendra, M., Jadliwala, M., Chandrasekaran, M. e Upadhyaya, S. (2005). Quantifying trust in mobile ad-hoc networks. *International Conference on Integration of Knowledge Intensive Multi-Agent Systems*.

Yu, B., Singh, M. e Sycara, K. (2004). Developing trust in large-scale peer-to-peer systems. *First IEEE Symposium on Multi-Agent Security and Survivability*.

Zhao, S., L. V. e GauthierDickey, C. (2005). Result verification and trust-based scheduling in peer-to-peer grids. *Fifth IEEE International Conference on Peer-to-Peer Computing*.

Índice por Autor