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.

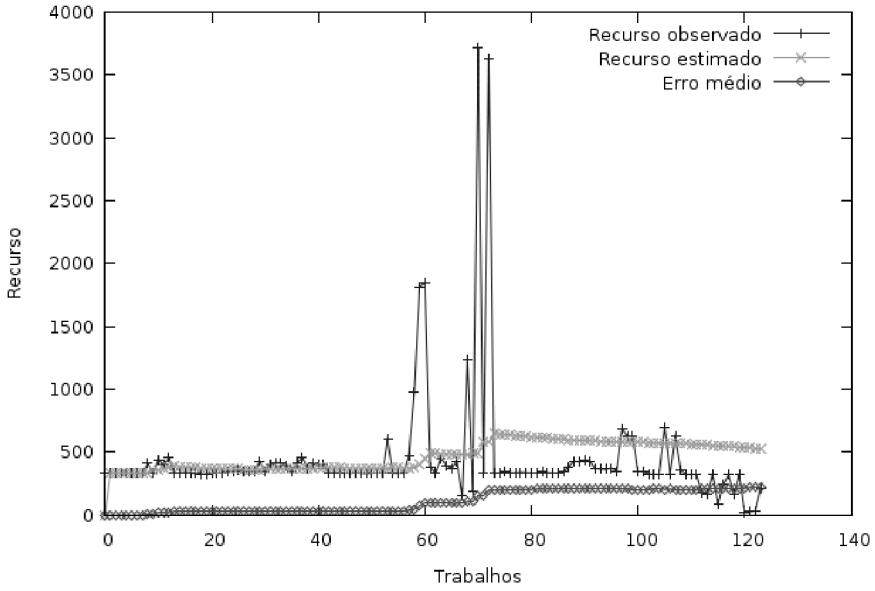

Figura 1. Previsão do tempo de execução de uma aplicação utilizando intervalo de confiança

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.

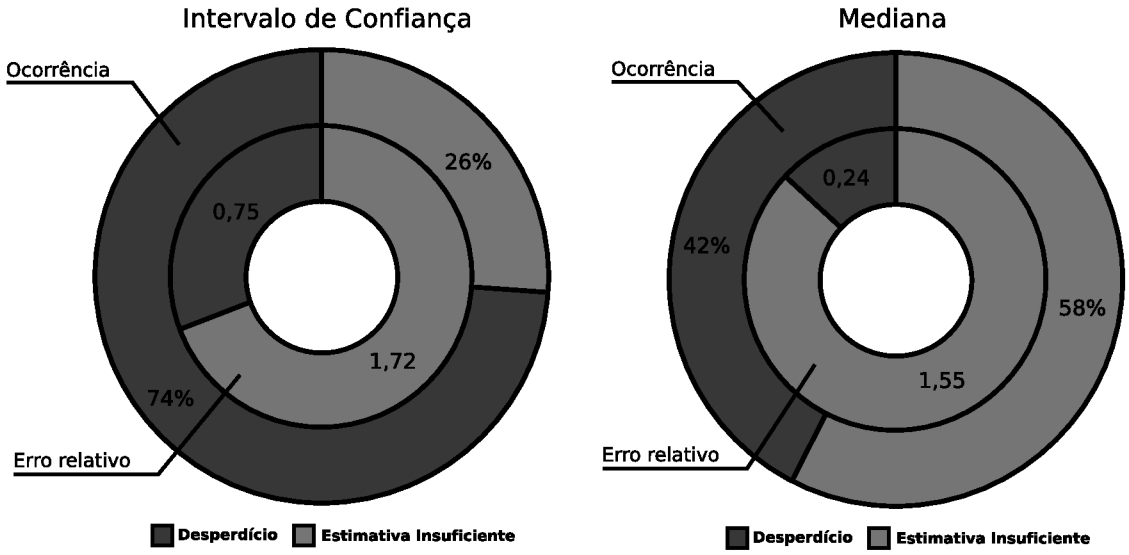

Figura 2. Proporção de desperdícios e estimativas insuficientes

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

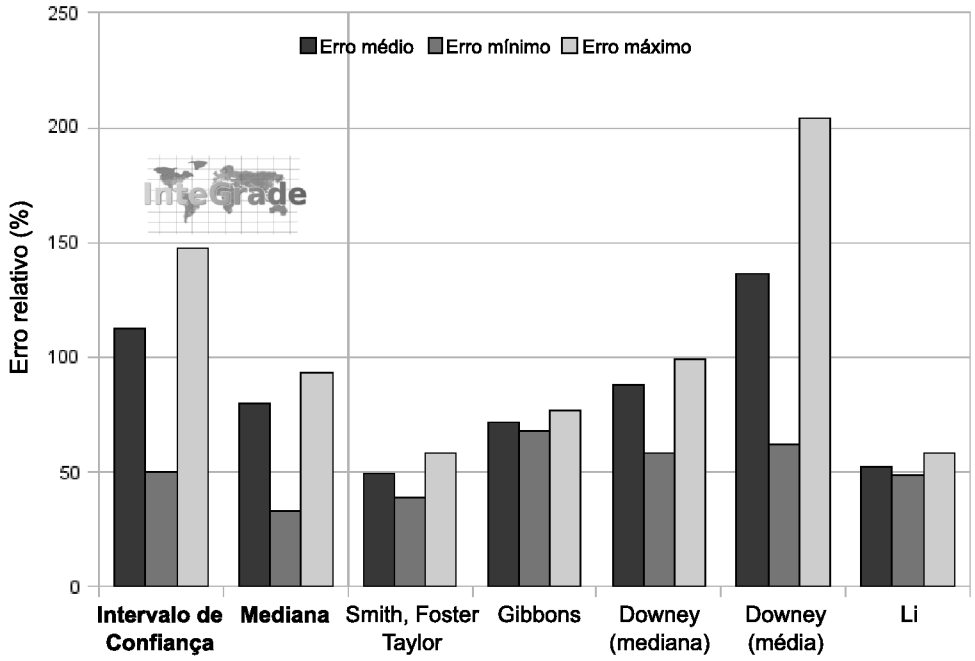

Figura 3. Gráfico comparativo com outros trabalhos da literatura

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.