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.

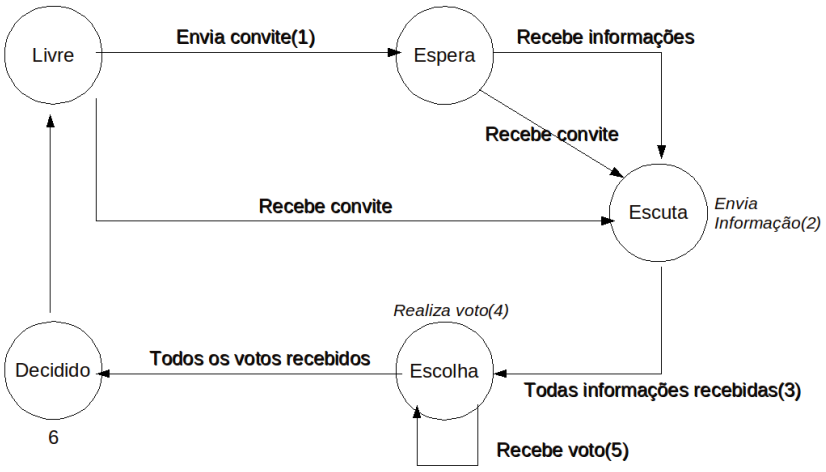

Figura 1. Diagrama de estados do scheduler durante o consenso

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.

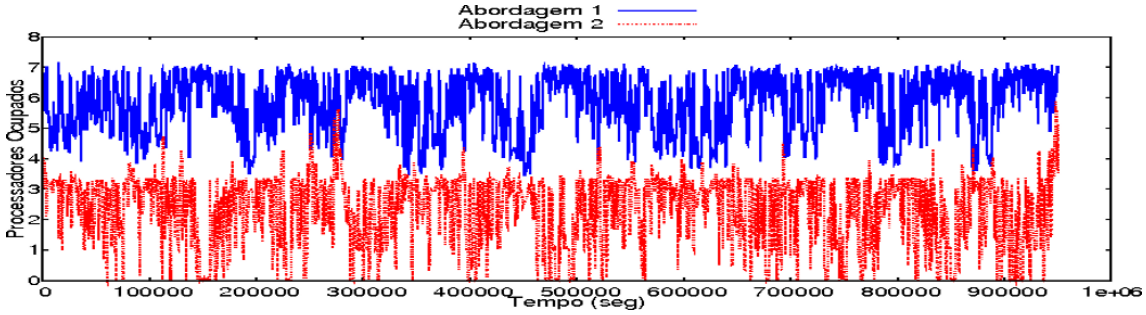

Figura 2. Processadores ocupados, 5 escalonadores

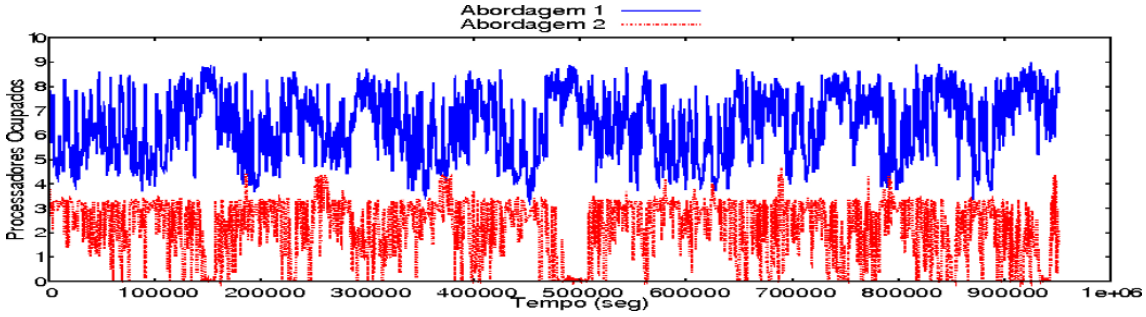

Figura 3. Processadores ocupados, 10 escalonadores

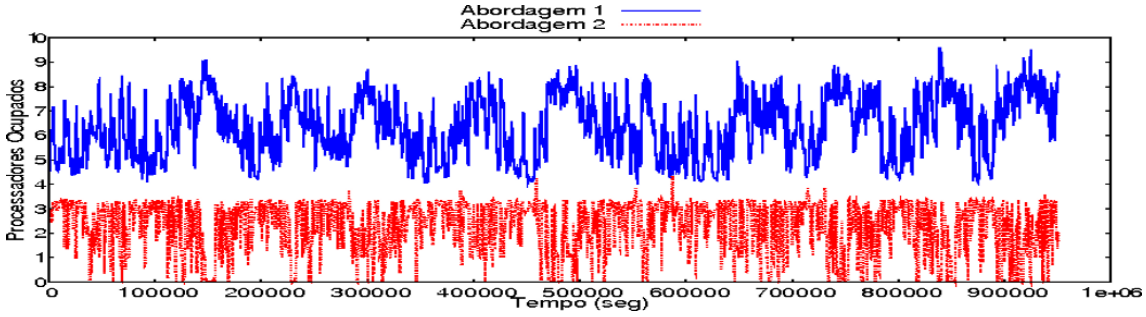

Figura 4. Processadores ocupados, 50 escalonadores

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.