Um Serviço Escalável e Robusto para Gerenciamento de Membros em Grades Computacionais de Grande Escala *

Fernando Castor-Filho[1], Rodrigo Castro[2], Augusta Marques[2], Francisco M. Soares-Neto[2], Raphael Y. de Camargo[3], Fabio Kon[4]

[1]Universidade Federal de Pernambuco (UFPE)
[2]Universidade de Pernambuco (UPE)
[3]Universidade Federal do ABC (UFABC)
[4]Universidade de São Paulo (USP)

castor@cin.ufpe.br, {rmcsc,armf,fmssn}@dsc.upe.br
raphael.camargo@ufabc.edu.br, kon@ime.usp.br

Abstract. *To avoid wasting resources, computational grids must be capable of consistently detecting node failures quickly and accurately. This paper proposes a decentralized group membership service that provides grid members with a consistent view of its current status. This service includes a set of mechanisms that make it capable of tolerating a large number of simultaneous node failures. Also, it assumes that failed nodes can become correct again and disseminates information about membership changes in a scalable and efficient manner. Preliminary evaluation results have shown that the service is scalable and robust both in the absence and in the presence of node failures.*

Resumo. *Para evitar desperdício de recursos, grades computacionais devem ser capazes de detectar consistentemente falhas de seus participantes de maneira rápida e precisa. Este artigo propõe um serviço descentralizado de gerenciamento de membros do grupo que permite que os membros da grade tenham uma visão consistente de sua situação atual. Este serviço inclui um conjunto de mecanismos que o tornam capaz de tolerar falhas simultâneas de um grande número de nós da grade. Adicionalmente, considera que nós falhos podem voltar a funcionar corretamente e dissemina informação sobre mudanças no conjunto de membros da grade de forma escalável e eficiente. Uma avaliação preliminar do serviço proposto mostrou que ele é escalável e robusto tanto na ausência quanto na presença de falhas de nós.*

1. Introdução

Apesar de todos os seus benefícios, tanto os já alcançados quanto os potenciais, a Computação em Grade [Foster and Kesselman 1999] ainda é uma área de pesquisa bastante ativa e com diversos problemas que precisam ser resolvidos. Um desses problemas ainda em aberto é tolerância a falhas. Com frequência, computações realizadas por grades duram muitos dias. Além disso, após se reservar recursos de uma grade, podem se passar horas, ou até mesmo dias, até que eles estejam disponíveis. Portanto, um defeito

*Este trabalho é financiado pelo CNPq/Brasil, processos #481147/2007-1 e #550895/2007-8.

de um nó da grade pode desperdiçar muitos dias de trabalho e exigir que os recursos sejam novamente reservados. Esse problema é particularmente grave no contexto das grades oportunistas [Goldchleger et al. 2004], cujas estações de trabalho podem ser compartilhadas com usuários locais e somente fazer parte da grade quando estiverem ociosas. Neste cenário, falhas são eventos corriqueiros, já que os recursos das máquinas da grade podem ser solicitados por seus usuários locais, o que faz com que a máquina seja temporariamente removida da grade. Tornar a grade tolerante a falhas pode poupar tempo e permitir que ela seja usada de forma mais eficiente.

Para se recuperar de defeitos de seus nós ou enlaces de comunicação, uma infraestrutura de grade precisa antes detectar esses defeitos e garantir que seus participantes estejam cientes deles e possam tomar as providências apropriadas. Além disso, grades devem ser capazes de lidar com entradas dinâmicas de processos, tanto novos quanto previamente falhos. Na literatura de sistemas distribuídos, o serviço de middleware responsável por essas tarefas é geralmente chamado de serviço de Gerenciamento de Membros do Grupo [Birman 1993], ou simplesmente de Serviço de Gerenciamento de Membros (*Group Membership Service* – GMS). Este trabalho parte do pressuposto que um GMS é uma parte tão importante para a construção de grades computacionais tolerantes a falhas quanto para qualquer outro sistema distribuído de grande escala. Mesmo não sendo a única opção, o GMS se torna uma excelente escolha, já que permite que os membros da grade tenham uma visão consistente sobre os processos desta. Isso torna a recuperação mais rápida, já que cada processo tem conhecimento sobre os nós que estão vivos e pode realizar a migração de um processo que estava rodando em um nó que falhou para um nó não-falho com mais rapidez. Além disso, para aplicações fortemente acopladas, é importante que todos os processos que dependem de um processo que falhou saibam disso o mais rápido possível, mesmo que não o estejam monitorando diretamente.

Neste artigo, são apresentados o projeto e a avaliação de um GMS que tem por objetivo ser prático e atender aos requisitos de grades computacionais de grande escala. A apresentação é organizada em duas partes. Na primeira, é apresentado um GMS ("GMS básico" ou "serviço básico") construído com o intuito de satisfazer os diversos requisitos impostos por grades computacionais tolerantes a falhas [Medeiros et al. 2003], como escalabilidade, eficiência, distribuição e facilidade de configuração [Castor-Filho et al. 2008]. O GMS básico (Seção 4) combina disseminação epidêmica de informação [Eugster et al. 2004] (ou baseada em boatos [Guerraoui and Rodrigues 2006]) e detectores cumulativos de defeitos [Hayashibara et al. 2004, Satzger et al. 2007] para prover um serviço capaz de satisfazer os requisitos listados anteriormente. Na segunda parte (Seção 5), são descritas algumas melhorias que tornam o serviço básico mais resistente a falhas de processos, enquanto mantêm-se as características desejáveis do serviço básico. Tais melhorias são a principal contribuição deste trabalho. O serviço proposto é avaliado através de vários experimentos (Seção 6) em termos de duas características fundamentais: escalabilidade e robustez. Esses experimentos mostraram que ele é, de fato, escalável e robusto, mesmo quando um grande número de falhas acontece em um curto espaço de tempo.

2. Preliminares

Esta seção descreve as premissas deste trabalho e faz uma rápida explanação sobre disseminação epidêmica de informação e detectores cumulativos de defeitos.

Premissas Básicas. Consideramos que uma grade é formada por um conjunto de processos que se comunicam por troca de mensagens. Não diferenciamos explicitamente processos, nós e máquinas na grade e usamos esses termos indistintamente. Também usamos o termo "membro" como um sinônimo para processo e nó. Consideramos que todo processo pode se comunicar diretamente com qualquer outro processo através da camada de transporte subjacente. Nenhuma suposição é feita sobre as aplicações que serão executadas na grade nem sobre os modelos de programação paralela que elas adotam.

Nós supomos um modelo de quebra-recuperação (*crash-recovery*) onde processos falham silenciosamente, mas podem se recuperar de defeitos e voltar à grade. Para grades oportunistas, esse modelo de falhas é mais apropriado que o modelo de quebra (*crash*) adotado pela maioria dos trabalhos sobre detecção de defeitos e GMS [Horita et al. 2005, Leitao et al. 2007]. Devido ao grande número de processos em uma grade, supomos que falhas são eventos comuns e existem processos falhando e voltando à grade o tempo todo. É até mesmo possível que uma grande quantidade (30% ou mais) dos processos da grade falhem simultaneamente devido a eventos como quedas de energia, partições da rede e *worms* na Internet [Hayashibara et al. 2004]. Consideramos que canais de comunicação não perdem mensagens e funcionam de forma parcialmente síncrona [Guerraoui and Rodrigues 2006].

Disseminação Epidêmica de Informação. Protocolos de disseminação epidêmica de informação (ou baseados em boatos) reproduzem a forma pela qual doenças contagiosas (ou boatos) espalham-se. Em um protocolo epidêmico, quando um processo p precisa disseminar alguma informação para outros processos, ele escolhe alguns aleatoriamente e envia a informação. Cada processo q que recebe a informação a combina com as informações mais atuais que possui (i.e. é infectado) e repete o procedimento, escolhendo aleatoriamente alguns processos e enviando a nova informação gerada.

O número de processos para os quais cada processo dissemina informações em cada rodada de um protocolo epidêmico é chamado de *fanout* do protocolo e é um de seus parâmetros mais importantes. Na prática, pequenos valores de *fanout* asseguram altos níveis de confiabilidade e velocidade de disseminação. Foi demonstrado, em estudos teóricos [Eugster et al. 2004], que a informação disseminada por um processo será recebida por todos os processos com alta probabilidade (quase 1) se cada processo repassá-la para aproximadamente $\log X + c$ processos, onde c é uma constante e X é o número de processos total do grupo. Muitos trabalhos já demonstraram, tanto na teoria quanto na prática, que protocolos epidêmicos são robustos e escaláveis [Das et al. 2002, Eugster et al. 2004, Horita et al. 2005, Leitao et al. 2007].

Detectores Cumulativos de Defeitos. Detectores de Cumulativos Defeitos [Hayashibara et al. 2004] são uma solução para a inerente falta de flexibilidade dos detectores de defeitos tradicionais baseados em *heartbeat*. Em um detector de defeitos tradicional, se um processo p monitora um processo q, o último envia periodicamente mensagens de *heartbeat* ao primeiro. Se, durante um certo período de tempo T_{to}, p não recebe nenhuma mensagem de *heartbeat* de q, q é considerado falho.

Detectores cumulativos de defeitos consideram que os tempos entre recebimentos consecutivos de *heartbeats* são amostras aderentes a uma distribuição de probabilidades e, considerando o tempo desde o último *heartbeat* recebido, calculam a probabilidade de

que um processo monitorado tenha falhado. Eles se adaptam naturalmente a mudanças nas condições da rede ao mesmo tempo que escondem completamente de seus usuários a noção de tempo. Como detectores cumulativos de defeitos fornecem valores de suspeita em uma escala contínua (ao invés de apenas declarar um processo como suspeito ou não), é possível definir estratégias distintas de recuperação de defeitos baseadas no nível de suspeita de que um processo falhou. Essa característica resulta em uma maior flexibilidade para os administradores de sistemas.

3. Trabalhos Relacionados

Existem muitas abordagens para tornar grades tolerantes a defeitos. Poucas, porém, focam em GMSs. Horita et al. [Horita et al. 2005] propuseram um GMS que tem como alvo grades computacionais e utiliza disseminação epidêmica de informação. Esse trabalho baseia-se fortemente no SWIM [Das et al. 2002], um GMS que separa seus mecanismos de detecção de defeitos e disseminação de informação sobre falhas e baseia-se em uma abordagem epidêmica. Horita et al. adaptaram o SWIM para usar conexões TCP, ao invés de datagramas UDP. Os autores argumentam que essa estratégia torna o seu GMS mais compatível com *firewalls* e NATs. O serviço proposto neste trabalho segue essa filosofia, mas inclui mecanismos que o tornam mais robusto na presença de falhas e tão escalável quanto o GMS de Horita et al.(Seção 6).

Jain e Shyamasundar [Jain and Shyamasundar 2004] descrevem um serviço de detecção de defeitos e grupo de processos para grades centrado no conceito de grupos de *heartbeats*. Grupos de *heartbeats* prescrevem uma organização hierárquica de detectores de defeitos que aumentam a escalabilidade, mas diminuem a confiabilidade, porque cada grupo é coordenado por um líder central (assistido por um único processo de *backup*). Essa abordagem é altamente vulnerável a falhas simultâneas do coordenador e do processo de *backup*, diferentemente do GMS proposto neste trabalho. Além disso, não usa disseminação epidêmica de informação. Shi et al. introduziram a arquitetura ALTER [Shi et al. 2005] para detecção de defeitos em grades. Grupos de processos são gerenciados por um serviço de indexação separado que funciona como um ponto único de falha para o sistema. Diferentemente desses trabalhos, o GMS proposto funciona de forma completamente descentralizada e é capaz de tolerar falhas simultâneas de grandes quantidades de nós. Além disso, não está claro o quão escalável ALTER é. O Globus HBM [Stelling et al. 1998] utiliza detectores de defeitos não confiáveis [Guerraoui and Rodrigues 2006] e distribuídos. Ele dissemina informações sobre processos defeituosos através de inundação e, por conseguinte, não é escalável. Além disso, não lida com questões relativas a gerenciamento de membros.

GRIDTS [Favarim et al. 2007] é uma infra-estrutura para computação em grade que usa memória compartilhada distribuída para comunicação entre os nós. Diversos mecanismos de tolerância a falhas são implementados por GRIDTS, como transações, pontos de recuperação e replicação, e a infra-estrutura não tem pontos únicos de falhas. Esse trabalho é complementar ao apresentado neste artigo, já que sua ênfase está nos mecanismos de recuperação de defeitos. O GMS proposto, por sua vez, lida principalmente com a detecção de defeitos e a disseminação de informação sobre estes.

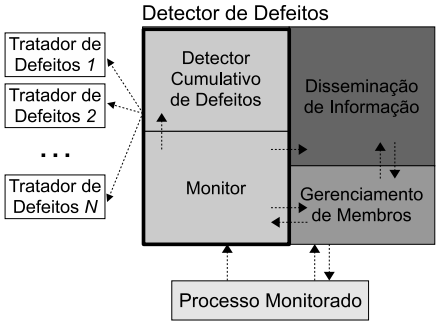

Figura 1. Principais componentes do GMS. Setas tracejadas indicam a direção de interação entre componentes

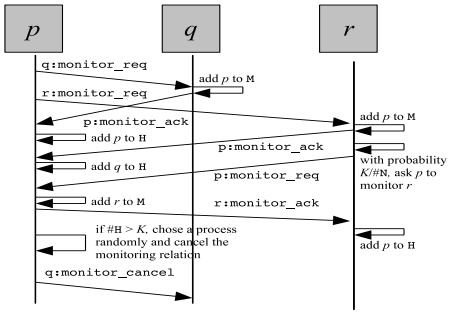

Figura 2. Estabelecimento e cancelamento de relações de monitoramento entre três processos.

4. Serviço Básico

Esta seção apresenta o serviço básico, descrito com mais detalhes em outro tabalho [Castor-Filho et al. 2008]. Esse GMS básico possui três componentes: (i) um detector de defeitos; (ii) um componente de disseminação de informação; e (iii) um componente de gerenciamento de membros. O serviço básico está implementado na linguagem Lua [Ierusalimschy et al. 1996], o que o torna leve, portátil e facilmente extensível. Para comunicação entre processos, foi usado um ORB CORBA leve chamado OiL [Maia et al. 2006], também escrito em Lua. A Figura 1 mostra uma arquitetura de alto nível do GMS que é executado em cada nó do grupo. No restante desta seção, cada um desses componentes é explicado.

O Componente de Gerenciamento de Membros mantém um registro dos processos que cada processo conhece, os que ele acredita que tenham falhado, os que o monitoram e os que ele monitora. Além do mais, esse componente coordena a interação entre o Detector de Defeitos e o componente de Disseminação de Informação. Cada processo em um grupo pode ser monitorado por no máximo outros K processos, i.e., valores maiores de K significam uma rede de monitoramento mais robusta. Configurar K com um inteiro pequeno, como 5 ou 6, é suficiente para maioria dos propósitos práticos [Horita et al. 2005, Leitao et al. 2007]. Se um processo r é monitorado por um processo s, há uma conexão TCP aberta entre os dois através da qual são enviadas mensagens de *heartbeat* e outros tipos de informação.

A Figura 2 ilustra o funcionamento do componente de gerenciamento de membros. Na figura, a notação `a:m` indica o envio de uma mensagem `m` para o processo `a`. O exemplo mostra um processo p solicitando que outros dois processos, q e r, o monitorem, através da mensagem `monitor_req`. Os dois processos respondem positivamente com mensagens `monitor_ack`. Após responder à solicitação de p, r também requisita que p o monitore. Depois que p recebe as respostas de q e r, percebe que está sendo monitorado por mais que K processos. Consequentemente, escolhe aleatoriamente um dos processos (no exemplo, o escolhido foi q) que o monitora e cancela essa relação de monitoramento (mensagem `monitor_cancel`). Esse procedimento mantém o número de relações de monitoramento próximo a K e introduz maior aleatoriedade no estabelecimento dessas relações. Na figura aparecem alguns conjuntos de informações mantidos pelo componente de gerenciamento de membros de cada processo. Para um processo arbitrário x,

(i) `N` é o conjunto de todos os processos que x conhece (ii) `M` é o conjunto de processos que x monitora (por exemplo, q inclui p em `M` quando recebe `monitor_req`) e (iii) `H` é o conjunto dos processos que monitoram x (p inclui r em `H` quando recebe `monitor_ack`). Para garantir a consistência das informações sobre o grupo de processos em situações nas quais processos podem falhar e se recuperar (usando o mesmo identificador), o serviço proposto inclui um mecanismo de épocas [Guerraoui and Rodrigues 2006]. Esse mecanismo garante que, se um processo falhar e posteriormente se recuperar, apenas as mensagens enviadas por ele após se recuperar são levadas em conta pelos outros processos.

O Componente Detector de Defeitos é formado pelos componentes Monitor e Detector Cumulativo de Defeitos (AFD) (Figura 1). O componente Monitor é responsável por coletar mensagens de *heartbeat* que processos monitorados enviam periodicamente aos processos monitores. O parâmetro T_{hb} determina o tempo entre dois *heartbeats* consecutivos enviados por um processo monitorado. No serviço proposto, processos monitorados escolhem aleatoriamente processos para monitorá-los, já que isso é fundamental para a confiabilidade de protocolos epidêmicos [Das et al. 2002, Eugster et al. 2004]. Entretanto, sempre enviam mensagens de *heartbeat* para os mesmos processos, enquanto durar o relacionamento de monitoramento. Essa abordagem garante que o AFD coleta informações suficientes sobre os processos monitorados.

Ao receber um *heartbeat*, o componente Monitor calcula o tempo decorrido desde o recebimento do *heartbeat* anterior e passa essa informação ao componenente AFD. O AFD registra as últimas NS amostras que ele recebeu, onde NS é um parâmetro definido pelo usuário. Similarmente a outros trabalhos sobre detectores cumulativos de defeitos [Hayashibara et al. 2004, Satzger et al. 2007], NS foi configurado com valor 1000. A cada pulso do relógio, o Monitor pede ao AFD para procurar por processos falhos. AFDs calculam valores de suspeita (probabilidades de que processos monitorados tenham falhado) considerando que quanto maior é o número de *heartbeats* que levaram menos tempo para chegar do que o tempo decorrido desde que o último *heartbeat* foi recebido, maior a probabilidade de que o próximo *heartbeat* não chegue [Satzger et al. 2007]. Foi usada uma implementação otimizada para calcular a probabilidade de defeito usando no máximo $O(\log NS)$ comparações no pior caso, o que garante que valores de suspeita podem ser calculados rapidamente mesmo que NS seja muito grande.

O componente AFD pode ser configurado com diversos mecanismos de tratamento de defeitos. Estes são associados a limiares que definem a probabilidade de falha de um processo. Quando a suspeita de que um processo p falhou atinge um dos limiares estabelecidos, o mecanismo de tratamento de defeitos correspondente é disparado pelo AFD. Note que o AFD continua monitorando o processo p. Logo, se a suspeita de que p falhou atingir um limiar mais elevado, o AFD irá disparar o mecanismo de tratamento de falha correspondente e continuará a monitorá-lo, até que um dos mecanismos de tratamento remova p da lista de monitoramento do componente AFD. Com essa abordagem, ações preventivas de recuperação de defeitos podem ser definidas para limiares menores, enquanto procedimentos mais enérgicos podem ser definidos para os maiores.

O Componente de Disseminação de Informação notifica outros processos sobre mudanças no conjunto de processos associados e, mais geralmente, sobre outros processos do grupo. Em uma grade, informações sobre processos falhos devem ser disseminadas rapidamente, de modo a não impedir o progresso e evitar defeitos de aplicações, ou, em

casos onde defeitos de aplicações são inevitáveis, permitir que um trabalho possa ser reiniciado (ou reagendado) o mais rápido possível. Em contrapartida, informações sobre novos membros não são tão urgentes, já que não conhecer imediatamente os processos "novos" da grade não impede o progresso das aplicações rodando nela.

Para disseminar informações sobre defeitos, o GMS básico usa uma abordagem reativa e explícita. Isso significa que quando um processo descobre que um outro processo falhou, ele automáticamente envia essa informação a outros J processos que ele monitora ou que o monitoram. O parâmetro J determina o *fanout* da disseminação (Seção 2). Por outro lado, nenhuma ação explícita é tomada para disseminar informação sobre novos processos. Cada *heartbeart* que um processo p envia para um processo q inclui identificadores de alguns processos escolhidos aleatoriamente entre os que p conhece.

5. Um Serviço de Grupo de Processos Mais Robusto

A Seção 4 apresentou as características fundamentais do GMS proposto. Nesta seção são descritas duas melhorias que foram aplicadas ao serviço básico para torná-lo mais robusto. Tais melhorias funcionam como elementos acessórios do serviço e podem ser ligadas e desligadas.

5.1. Garantindo um Mínimo de Processos Monitores

Quando um processo p detecta que um processo r que o monitora falhou, p imediatamente solicita que um outro processo s, escolhido aleatoriamente, o monitore. Caso s também tenha falhado, p não tenta novamente de imediato pois isso poderia resultar em um grande número de mensagens adicionais sendo injetadas na rede, caso muitas falhas ocorressem em um curto espaço de tempo. O serviço proposto inclui um mecanismo simples e econômico para garantir que cada processo seja sempre monitorado por pelo menos K outros processos, garantindo assim a robustez do grupo. O pseudo-código apresentado a seguir ilustra o funcionamento desse mecanismo. O operador $\#$ retorna a cardinalidade de um conjunto.

```
-- Cada processo p, a cada T_hb segundos, executa o código abaixo:
if #N > K and (#H + #A) < K then
 local selected_proc = get_monitoring_candidate()
 if should_request_monitoring() then request_monitoring(selected_proc) end
end
```

Esse mecanismo utiliza três conjuntos de processos mantidos por cada processo: `N` e `H`, descritos na Seção 4, e `A`, o conjunto de processos para os quais p enviou solicitações de monitoramento que ainda não foram respondidas. Periodicamente p verifica se `#H` $< K$. Se for o caso, houver um número suficientemente grande de processos na grade (`#N` $> K$) e `#H + #A` $< K$, p realiza um sorteio (chamada a `should_request_monitoring()`) para decidir se vai ou não solicitar que um processo q (`selected_proc`) escolhido aleatoriamente o monitore. É necessário que `#N` seja maior que K porque p está incluso em `N` e consideramos que não faz sentido um processo monitorar a si mesmo. Além disso, o número de requisições de monitoramento pendentes é levado em conta na hora de um processo decidir se vai ou não enviar novas requisições. Dessa forma evita-se a criação de novas relações de monitoramento que precisam ser canceladas logo após sua criação. Se p realizar a solicitação de monitoramento, a requisição é enviada e q é incluído em `A`. Essa abordagem aumenta consideravelmente a

confiabilidade do protocolo a um custo baixo em termos de número de mensagens extras (Seção 6).

5.2. *Heartbeats* Informativos

O GMS proposto dissemina informação sobre novos processos de maneira periódica, enviando essa informação de carona em mensagens de *heartbeat*. Como o envio de *heartbeats* adere à abordagem epidêmica, se K, o *fanout* (Seção 2) do protocolo, for suficientemente grande, todos os processos do grupo receberão os *heartbeats* com alta probabilidade em um pequeno número de turnos (da ordem de $\log \#\mathtt{N}$). O serviço proposto não garante, porém, que todos os processos monitoram outros processos, embora garanta que todos são monitorados. Logo, embora seja improvável, é possível que haja processos que nunca recebem informações sobre novos processos por nunca receberem *heartbeats*.

Para solucionar esse problema, foi adotada uma solução simples que não modifica o funcionamento do serviço básico. A cada T_{hb} segundos, cada processo envia uma mensagem de *heartbeat* para um dos processos que ele monitora, ou seja, na direção oposta à do fluxo de *heartbeats*. Esse *heartbeat* não é utilizado para fins de detecção de defeitos e sua função é estritamente informativa, i.e., um processo não suspeitará que o outro falhou por não receber nenhuma dessas mensagens por um longo período. O processo para o qual o *heartbeat* informativo (HBI) é enviado é escolhido aleatoriamente. Como cada processo é monitorado por K outros processos, essa abordagem faz com que, na média, cada processo receba um HBI a cada T_{hb} segundos.

O uso de HBIs resulta em um custo constante por processo, em termos de número de mensagens enviadas, a cada turno do protocolo. Para valores baixos de K, esse custo pode ser significativo, e.g., se $K = 4$, enviar uma mensagem a mais significa aumentar em 25% o número total de mensagens enviadas por turno. Para valores mais altos ($K > 10$), porém, o impacto desse mecanismo extra é pequeno. Adicionalmente, ele pode ser configurado para que os HBIs sejam enviados com menos frequência, por exemplo, a cada $10 * T_{hb}$ segundos, o que diminui consideravelmente o número de mensagens.

6. Avaliação

A avaliação do GMS proposto consistiu na execução de várias intâncias do serviço (de 40 a 200) em sete computadores com processadores Pentium IV de 2,53 GHz e com 1 GB RAM cada, rodando Kubuntu Linux 8.04 e comunicando-se por uma rede local *Ethernet* de 100 Mbps. Além disso, foi emulada uma rede de grande área usando o emulador WANem [Tata Consulting Services 2008]. WANem é um emulador paramétrico que dá suporte à configuração de muitos parâmetros de uma rede de grande área, como latência, *jitter*, % de duplicatas, perda de mensagens, etc. Dois desses parâmetros, latência e *jitter*, foram configurados para 500 ms e 250 ms, respectivamente. Em todos os experimentos realizados, todo o tráfego entre máquinas distintas passou pelo WANem, simulando, portanto, uma grade formada por até sete aglomerados separados pela Internet. Cada máquina executou de 20 a 30 instâncias do serviço. Para cada instância, foram usadas as seguintes configurações: T_{hb} = 2s, K = 4, e J = 6 (Seção 4).

O objetivo principal dos experimentos foi avaliar a robustez e a escalabilidade do serviço proposto. Em cada experimento, instâncias do serviço foram iniciadas a intervalos regulares (2 segundos de diferença entre duas instâncias consecutivas). Muitas instâncias

do serviço de grupo de processos receberam informação sobre apenas um processo existente. Além disso, mais de 20% dos membros de cada aglomerado também receberam informações sobre um único nó localizado em um aglomerado diferente. Cada experimento foi executado pelo menos três vezes e os resultados apresentados correspondem às médias dessas três execuções. Em alguns dos experimentos, comparamos diferentes configurações do serviço proposto (com ambos os mecanismos descritos nas Seções 5.1 e 5.2 ligados, com ambos desligados e com apenas um deles ligado). Também foi utilizado em alguns experimentos o trabalho de Horita et al. [Horita et al. 2005]. A implementação do serviço de grupo de processos de Horita et al. foi realizada pelos autores deste trabalho, de forma independente do serviço proposto.

6.1. Robustez

A robustez do GMS foi avaliada principalmente em termos de sua capacidade de criar novas relações de monitoramento quando falhas catastróficas ocorrem, de modo a manter, para todos os processos, o invariante #H + #A = K (Seção 5.1). Como o serviço proposto visa garantir que os membros do grupo tenham uma visão consistente, falhas de grandes números de processos não devem resultar em processos isolados ou monitorados por menos que K outros processos. A Figura 3(a) mostra que o serviço proposto, quando combinado com o mecanismo descrito na Seção 5.1, é de fato robusto. O gráfico mostra que, devido à ocorrência de falhas (100 no total, introduzidas em dois blocos, conforme descrito na Seção 6.2), o número médio de processos monitorando cada processo cai com o passar do tempo quando se usa os serviços básico e de Horita et al. Em contrapartida, a abordagem proposta garante que, rapidamente após a ocorrência de um grande número de falhas, o número de processos monitorando cada processo volta a ser igual a K. Ao final de 15000 turnos de troca de mensagens, o valor médio de #H permanece igual a 4 ($K = 4$), enquanto esse valor é 3,55 quando se usa o serviço básico e 3,7 quando a abordagem de Horita et al. é empregada. Mais especificamente, enquanto 100% dos processos mantiveram #H = 4 quando o GMS proposto foi usado, esse número caiu para 72 e 61%, quando foram empregados os serviços de Horita et al. e básico, respectivamente. Nestes dois últimos casos, 26 e 33% dos processos terminaram o experimento sendo monitorados por 3 processos, enquanto 2 e 6%, terminaram com #H = 2. Adicionalmente, para o serviço proposto, o valor de #H voltou a ser igual a K para os processos corretos em média 40 turnos depois das falhas serem introduzidas, o que significa que o grupo voltou à sua situação ideal depois de apenas duas rodadas de envio de *heartbeats* (aprox. 4s).

O gráfico da Figura 3(b) mostra o número médio de mensagens enviadas por processo a cada 1000 turnos. Em um cenário sem falhas, idealmente, esse número deve se manter constante. Quando falhas ocorrerem, depois de oscilar um pouco, ele deve voltar rapidamente ao seu valor original. O gráfico mostra que isso não ocorre nos serviços básico e de Horita et al.; o número médio de mensagens enviadas cai após cada falha, devido ao fato de nem todos os processos conseguirem reestabelecer o valor de #H, conforme evidenciado no gráfico da Figura 3(a). Esse fenômeno fica evidente nos últimos 1000 turnos do experimento, onde o número médio de mensagens enviadas por processo usando a abordagem descrita na Seção 5.1 permanece constante com relação aos intervalos anteriores onde falhas não ocorreram, enquanto os serviços básico e de Horita et al. apresentam reduções de 12,86 e 7,55%, respectivamente.

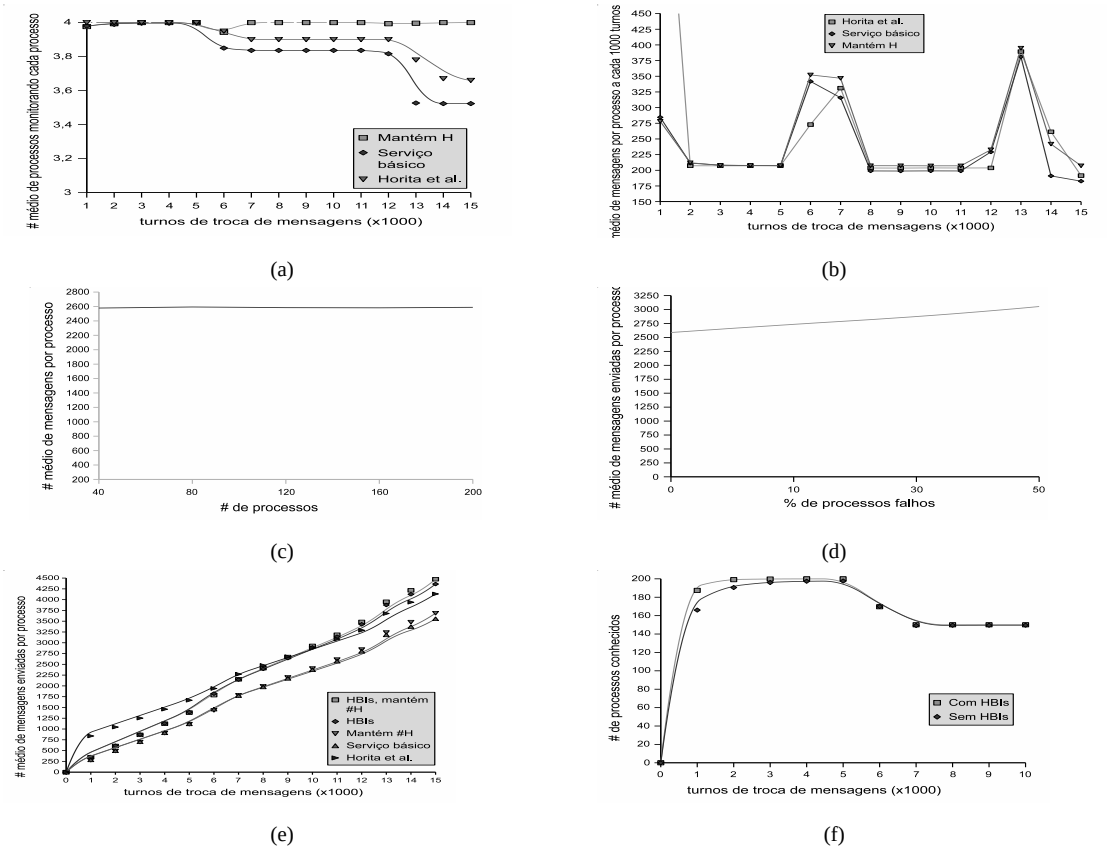

Figura 3. Avaliação da escalabilidade e da robustez do GMS

6.2. Escalabilidade

A Figura 3(c) apresenta os resultados dos experimentos onde nenhuma falha foi injetada. Foram realizados experimentos com 40, 80, 120, 160 e 200 instâncias do serviço e cada experimento envolveu 10000 turnos de troca de mensagens. Neste experimento, o serviço foi executado com todos os mecanismos adicionais ligados. O gráfico mostra que o número médio de mensagens enviadas por processo mantém-se praticamente constante com o aumento do número de processos. A maior média de mensagens enviadas por processo (nos experimentos envolvendo 80 processos) foi apenas 0,56% maior que a menor média de mensagens enviadas por processo (40 processos). Com o passar dos experimentos, o número de mensagens enviadas por processo não exibiu uma tendência a crescer. A Figura 3(d) mostra como o número médio de mensagens varia de acordo com o número de processos falhos, em uma grade com 200 nós. As falhas foram injetadas todas ao mesmo tempo. Para um número considerável de defeitos (10% dos processos da grade), a quantidade média de mensagens enviadas por processo cresceu 4,4% quando comparado com o cenário onde nenhum defeito ocorreu. Quando um número muito grande de falhas é injetado (50% dos processos), o número de mensagens cresce 11,79%. Esses resultados sugerem que o serviço proposto é escalável na presença e na ausência de defeitos.

O gráfico da Figura 3(e) mostra o efeito que falhas de processos causam no número total médio de mensagens enviadas por instância do serviço com o passar do tempo, em uma grade com 200 nós. O gráfico mostra os resultados para as quatro variantes descritas anteriormente, além do serviço proposto por Horita et al. Neste experimento, foram in-

jetadas falhas em 50% dos processos rodando na grade. Essas falhas foram injetadas em duas etapas, com 25% dos processos falhando em cada uma delas, a fim de verificar se o serviço proposto seria capaz de se manter consistente frente a falhas catastróficas reincidentes. Os dois grupos de falhas foram injetados entre os turnos 5000 e 6000 e 12000 e 13000. O gráfico evidencia o custo adicional imposto pelos mecanismos descritos na Seção 5. A versão mais completa do serviço proposto (HBIs, Mantém #H) incorreu em um número de mensagens 8,27% maior que o do serviço de Horita et al. e 25,8% maior que o do serviço básico. A maior parte desse *overhead* é causada pelo envio de HBIs. Desligar o mecanismo responsável por manter constante o número de processos monitorados (Mantém #H) resulta em uma redução de apenas 2,5% no número de mensagens.

Apesar do custo de usar HBIs, esse mecanismo é útil quando novos processos precisam ser conhecidos rapidamente pelos processos existentes. O gráfico da Figura 3(f) mostra a evolução no número médio de processos conhecidos pelos membros do grupo quando HBIs são usados e quando eles não são. A região em que o número de processos conhecidos cai abruptamente é aquela onde falhas são introduzidas. Na abordagem sem HBIs, depois de decorridos mais de 2000 turnos de troca de mensagens, na média cada processo conhece 95% dos membros, e.g., desconhece 1 em cada 20 membros. Já na abordagem com HBIs, passado o mesmo número de turnos, cada processo conhece, em média, 99% dos membros.

7. Conclusão

Foi apresentada uma proposta para um GMS que combina avanços recentes na disseminação de informação baseada em boatos e nos detectores cumulativos de defeitos a fim de produzir um serviço escalável e robusto. Adicionalmente, introduziu-se um mecanismo capaz de manter o conjunto de relações de monitoramento do GMS proposto estável mesmo quando um grande número de falhas ocorrem em um curto espaço de tempo. Até onde nós conhecemos, somos os primeiros a projetar e implementar um serviço de detecção de defeitos cumulativo onde os usuários podem associar diferentes mecanismos de tratamento de defeitos a diferentes limiares de defeitos.

Atualmente, o GMS apresentado neste trabalho está sendo incorporado à plataforma InteGrade [Goldchleger et al. 2004] para computação em grade. Uma vez terminada esta etapa, pretendemos realizar experimentos em cenários reais, envolvendo grades geograficamente distribuídas, a fim de avaliar a adequação do serviço proposto, tanto do ponto de vista de desenvolvimento quanto de execução. Adicionalmente, pretendemos tornar o GMS proposto mais dinâmico, capaz de configurar alguns de seus parâmetros automaticamente, com base nas características da rede e dos membros do grupo.

Referências

Birman, K. (1993). The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53.

Castor-Filho, F., Marques, A., de Camargo, R. Y., and Kon, F. (2008). A group membership service for large-scale grids. In *Proceedings of the 6th Workshop on Middleware for Grid Computing*, page 3, Leuven, Belgium.

Das, A., Gupta, I., and Motivala, A. (2002). Swim: Scalable weakly-consistent infection-style process group membership protocol. In *Proc. of the 32nd International Conf. on Dependable Systems and Networks*, pages 303–312.

Eugster, P. T., Guerraoui, R., Kermarrec, A.-M., and Massoulie, L. (2004). Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67.

Favarim, F., da Silva Fraga, J., Lung, L. C., and Correia, M. (2007). GRIDTS: A new approach for fault-tolerant scheduling in grid computing. In *Proc. 6th IEEE Int. Symposium on Network Computing and Applications*, pages 187–194, Cambridge, USA.

Foster, I. and Kesselman, C., editors (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco.

Goldchleger, A. et al. (2004). Integrade: Object-oriented middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16(8):449–459.

Guerraoui, R. and Rodrigues, L. (2006). *Introduction to Reliable Distributed Programming*. Springer-Verlag.

Hayashibara, N. et al. (2004). The phi accrual failure detector. In *Proc. of the 23rd International Symposium on Reliable Distributed Systems*, pages 66–78.

Horita, Y., Taura, K., and Chikayama, T. (2005). A scalable and efficient self-organizing failure detector for grid applications. In *Proceedings of the 6th ACM/IEEE International Workshop on Grid Computing*.

Ierusalimschy, R., de Figueiredo, L. H., and Filho, W. C. (1996). Lua - an extensible extension language. *Software: Practice Experience*, 26(6):635–652.

Jain, A. and Shyamasundar, R. (2004). Failure detection and membership management in grid environments. In *5th International Workshop on Grid Computing*.

Leitao, J., Pereira, J., and Rodrigues, L. (2007). Hyparview: A membership protocol for reliable gossip-based broadcast. In *Proc. of the 37th International Conf. on Dependable Systems and Networks*.

Maia, R., Cerqueira, R., and Cosme, R. (2006). OiL: An object request broker in the Lua language. In *Proc. 24th Brazilian Symposium on Computer Networks*.

Medeiros, R., Cirne, W., Brasileiro, F. V., and Sauvé, J. P. (2003). Faults in grids: Why are they so bad and what can be done about it? In *Proceedings of the 4th IEEE International Workshop on Grid Computing (GRID 2003)*, pages 18–24, Phoenix, USA.

Satzger, B. et al. (2007). A new adaptive accrual failure detector for dependable distributed systems. In *Proc. of the 22nd ACM Symposium on Applied Computing*.

Shi, X. et al. (2005). Alter: Adaptive failure detection services for grids. In *Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 355–358.

Stelling, P., Foster, I. T., Kesselman, C., Lee, C. A., and von Laszewski, G. (1998). A fault detection service for wide area distributed computations. In *Proc. of the 7th Int. Symposium on High Performance Distributed Computing*, pages 268–279.

Tata Consulting Services (2008). Wanem - the wide area network simulator. Last visit: August 2008 – http://wanem.sourceforge.net/.