Análise do impacto de detectores de falha adaptativos no OurGrid

Abmar Grangeiro de Barros[1], **Francisco Vilar Brasileiro**[1]

[1] Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Laboratório de Sistemas Distribuídos
Av. Aprígio Veloso, s/n, Bloco CO, Bodocongó
58.429-900, Campina Grande, PB

{abmar,fubica}@lsd.ufcg.edu.br

Abstract. *Failure detectors are fundamental building blocks for implementing distributed systems. In this context, the state-of-the-art presents a lot of mechanisms that provide scalability, adaptation, flexibility and quality of service enforcement. Despite that, few systems in production actually use these mechanisms. We believe that one of the main reasons for this state of affairs is that the benefits of using a sophisticated failure detection service are not clearly understood. This paper presents a preliminary evaluation on the impact of adaptive failure detectors in distributed systems, taking the OurGrid, a middleware for grid computing, as an use case. We have analyzed, via simulation, the effect of using three of the most known adaptive mechanisms in literature on the task makespan. Our results show that the failure detection mechanism currently implemented in OurGrid performs substantially worse than any of the adaptive detectors analyzed.*

Resumo. *Detectores de falha são blocos fundamentais na implementação de sistemas distribuídos. Nesse contexto, o estado da arte apresenta uma série de mecanismos que viabilizam escalabilidade, adaptação, uso flexível e garantias de qualidade de serviço. Apesar disso, poucos sistemas em produção utilizam de fato esses mecanismos sofisticados. Acredita-se que uma das principais causas dessa estagnação no estado da prática é que os benefícios da utilização de mecanismos de detecção de falhas sofisticados não são claramente entendidos. Esse trabalho apresenta uma avaliação preliminar do impacto da utilização de detectores adaptativos em sistemas distribuídos, utilizando como caso de uso o middleware de computação em grade OurGrid. Analisou-se por meio de simulação o efeito do uso de três dos detectores adaptativos mais conhecidos na literatura no makespan das tarefas submetidas à grade. Os resultados das simulações comprovam que, ao considerar a métrica em questão, o detector de falhas estático atualmente implementado no OurGrid tem um desempenho significativamente pior do que todos os detectores adaptativos analisados.*

1. Introdução

Detectores de falha são oráculos provedores de informação acerca de falhas de processos, sendo reconhecidos como blocos fundamentais na construção de sistemas distribuídos.

Particularmente, detectores de falha não confiáveis [Chandra and Toueg 1996] representam um dos mais importantes resultados no contexto de sistemas assíncronos. São chamados de não confiáveis pois podem cometer erros, eventualmente suspeitando de processos corretos. Apesar disso, eles constituem uma abordagem modularizada para encapsular a sincronia necessária para contornar a impossibilidade de resolver o problema do consenso distribuído sujeito a falhas [Fischer et al. 1985], que consiste basicamente na impossibilidade de determinar, em um sistema totalmente assíncrono, se um processo falhou ou se está simplesmente lento.

Seguindo os primeiros resultados teóricos de Chandra, Hadzilacos e Toueg [Chandra and Toueg 1996, Chandra et al. 1996], várias implementações de detectores de falha não confiáveis foram propostas [Felber et al. 1999, Larrea et al. 2000, Gupta et al. 2001, Nunes and Jansch-Pôrto 2002] para resolver diferentes problemas clássicos em sistemas distribuídos. Além disso, um série de trabalhos foram dedicados à extensão do trabalho teórico inicial no sentido de adicionar recursos mais sofisticados ao serviço de detecção de falha, como escalabilidade [Gupta et al. 2001, Défago et al. 2003], adaptabilidade [Bertier et al. 2002, Chen et al. 2000, Hayashibara et al. 2004, Xiong and Defago 2007], flexibilidade [Hayashibara et al. 2004, Xiong and Defago 2007] e a possibilidade de medição e garantia de qualidade de serviço (QoS, do inglês *Quality of Service*) por meio da definição de métricas [Chen et al. 2000] para esse fim.

Apesar dos avanços já discutidos na área de detecção de falhas, poucos sistemas distribuídos em produção fazem uso das tecnologias sofisticadas desenvolvidas até então. Em vez disso, a grande maioria implementa mecanismos triviais, até mesmo baseados nas tecnologias de comunicação subjacentes (ex.: os tempos de detecção de RMI ou de *sockets* TCP/IP), sem levar em consideração escalabilidade, adaptabilidade ou flexibilidade.

As poucas aplicações e plataformas que implementam esse tipo de detecção, como o Cassandra [Lakshman and Malik 2010] e o ZooKeeper [Junqueira and Reed 2009, Barros 2010], ambos da Apache Software Foundation, e alguns sistemas históricos, como algumas implementações do FT-Corba [Su et al. 2001, Lung et al. 2006], carecem de avaliação ou não analisam os resultados em termos da aplicação suprajacente. Da mesma forma, outros trabalhos [Zhuang et al. 2003, Falai and Bondavalli 2005, Hayashibara et al. 2004] avaliaram a performance dos detectores de forma isolada da aplicação, utilizando as métricas específicas propostas por Chen et. al [Chen et al. 2000] ou relacionadas à camada de rede.

Assim, acredita-se que uma das principais causas dessa estagnação no estado da prática é que os benefícios para a aplicação da adoção de mecanismos de detecção de falhas sofisticados não são claramente entendidos. Além disso, uma vez que não há soluções de prateleira que implementam essas funcionalidades avançadas, desenvolvedores teriam que lidar com os custos de implementar tais soluções por completo.

Esse trabalho tem como objetivo analisar o impacto da adição de detectores de falha sofisticados no *middleware* de grades computacionais OurGrid [Cirne et al. 2006] por meio de simulação. O simulador OurSim [Fraga 2010], devidamente modificado para tratar os atrasos relacionados a falhas, será utilizado para esse fim. Será feita uma análise da variação do *makespan* das tarefas submetidas com a utilização dos diferentes mecanis-

mos adaptativos citados.

Dessa forma, pretende-se dar um passo preliminar no sentido de esclarecer os principais benefícios da utilização desses mecanismos, numa tentativa de aproximar o estado da arte ao estado da prática na área de detectores de falha em sistemas distribuídos

2. Detectores de falha não confiáveis

Boa parte do sucesso da abordagem de detectores de falha não confiáveis proposta por Chandra e Toueg [Chandra and Toueg 1996, Chandra et al. 1996] no projeto e implementação de sistemas distribuídos se baseia no fato de que esta permite uma separação elegante de interesses. Os oráculos são definidos em termos de propriedades abstratas ao invés de implementações específicas. Essas propriedades estão relacionadas a 1) completude (capacidade de detectar todas as falhas ocorridas) e 2) acurácia (capacidade de detectar apenas falhas reais). Assim, os protocolos podem ser projetados considerando tais propriedades, sem considerar suas implementações concretas [Chandra and Toueg 1996]. De um ponto de vista teórico, é mais fácil provar a corretude das aplicações que usam tal oráculo, enquanto que de um ponto de vista prático, o detector de falhas é visto como uma "caixa preta" acessada por meio de uma interface predefinida e de semântica precisa, que, por sua vez, facilita a portabilidade das aplicações.

Como discutido na introdução, o trabalho inicial de Chandra e Toueg foi também estendido no sentido da sofisticação dos detectores de falhas por meio da adição de recursos avançados que possibilitam escalabilidade, adaptabilidade, flexibilidade e garantia de QoS.

3. Fundamentação

3.1. Funcionamento dos detectores de falha não confiáveis

Detectores de falhas não confiáveis são processos que coletam informações sobre falhas de componentes em um sistema. Normalmente mantêm uma lista de objetos monitorados e seus respectivados estados (suspeito ou ativo).

Esses mecanismos geralmente descobrem componentes falhos usando dois tipos de mensagem de controle: *ping* e *heartbeat*. *Ping* é a mensagem enviada do detector de falhas para o componente monitorado. O detector de falhas então espera uma mensagem de confirmação para cada *ping* enviado.

Heartbeat é a mensagem enviada periodicamente (ou como resposta ao *ping*, dependendo do modelo do detector) pelo componente monitorado ao detector de falhas para indicar que ainda está ativo [Felber et al. 1999]. Se um *heartbeat* não for recebido até um determinado *timeout* expirar, aquele componente é considerado suspeito.

Quanto ao envio de mensagens de controle, os detectores podem ser classificados entre *pull* e *push* [Felber et al. 1999]. Na abordagem *push* apenas as mensagens de *heartbeat* são utilizadas e os fluxos de controle de informação têm mesma direção. Na abordagem *pull*, as mensagens de *ping* são também utilizadas para recuperar informação sobre o estado do componente monitorado. O OurGrid utiliza a abordagem *pull* em seu detector de falhas.

3.2. Mecanismos adaptativos

A adaptabilidade é a habilidade do serviço de manter a QoS aceitável mesmo em face de mudanças no ambiente de execução. Os ambientes sobre os quais grandes sistemas distribuídos executam normalmente são muito suscetíveis a essas flutuações, que podem ser causadas por aumentos momentâneos no tráfego na rede ou na carga de CPU. Os principais trabalhos que estudam adaptabilidade são baseados no uso de tempos dinâmicos de detecção.

Uma série de mecanismos adaptativos foram propostos na literatura. Particularmente, aqueles propostos por Chen et al [Chen et al. 2000] e Bertier et al [Bertier et al. 2002] são frequentemente usados como patamar de comparação com outras soluções. Por outro lado, o mecanismo proposto por Hayashibara et al [Hayashibara et al. 2004] tem ganhado uma atenção considerável por sua flexibilidade, no sentido de dar à aplicação a possibilidade de decidir sobre a falha de um processo de acordo com seus requisitos específicos. Nessa seção são discutidas as principais características dos mecanismos mencionados.

Chen. Chen et al [Chen et al. 2000] propuseram um detector de falhas adaptativo que estima os tempos de chegada das mensagens de controle (*heartbeat*) baseado em uma janela de amostragem. Nesse mecanismo, sempre que um *heartbeat* é recebido, o tempo de sua chegada é armazenado na janela de amostragem, e o tempo de chegada estimado para o próximo *heartbeat* (*EA*) é atualizado. Essa estimativa é uma média normalizada dos tempos de recepção de *heartbeats* armazenados na janela de amostragem.

Baseado nessa estimativa é calculado o ponto de expiração para o objeto monitorado. O ponto de expiração é dado pela soma da estimativa *EA* com uma margem de segurança α, que é um parâmetro de entrada desse mecanismo. Assim, para determinar se um objeto monitorado falhou, o detector de falhas verifica se o ponto de expiração atual foi ultrapassado.

Bertier. O mecanismo de detecção de falhas proposto por Bertier et al [Bertier et al. 2002] funciona de forma similar ao mecanismo de Chen. A principal diferença está na computação da margem de segurança α, que no mecanismo proposto por Bertier é dinâmica, e é atualizada de acordo com a qualidade da estimativa *EA*.

A computação do α é baseada na estimativa de Jacobson [Paxson 2000], que leva em consideração o erro entre a última estimativa e o tempo real de chegada de um *heartbeat*. Assim, a estimativa do α envolve três parâmetros: γ, que representa a importância da nova estimativa com relação às anteriores; e os parâmetros β e ϕ, que permitem ponderar a estimativa de acordo com a variância do erro. Além disso, quando um *heartbeat* é recebido depois do esperado, um passo de moderação é adicionado ao próximo ponto de expiração. Esse passo de moderação é um intervalo estático e também é um parâmetro do detector proposto por Bertier.

φ **Accrual** Detectores de falha *accrual* [Hayashibara et al. 2004, Défago et al. 2005] descrevem uma diferente abstração: ao invés de entregarem a aplicação um valor booleano (falho/operacional), eles produzem um valor de suspeição numa escala contínua. Assim, esse paradigma permite a separação dos requisitos da aplicação do mecanismo de detecção de falhas.

O detector φ Accrual [Hayashibara et al. 2004] também armazena *heartbeats* numa janela de amostragem. Essa informação histórica é utilizada para determinar a distribuição dos tempos entre chegadas. Então, essa distribuição é usada para computar o valor corrente de φ, que representa o nível de suspeição para um objeto monitorado. Esse mecanismo assume que os tempos de inter-chegada seguem uma distribuição normal. Porém, em [Xiong and Defago 2007], a distribuição exponencial é indicada como um melhor modelo para os tempos entre chegadas dos *heartbeats*.

Com o fim de utilizar o φ Accrual como um detector de natureza booleana, é necessário definir um limiar para o valor de φ, que, quando excedido, indica que o objeto monitorado é suspeito de falha.

3.3. OurGrid

O OurGrid é um *middleware* de computação em grades P2P que dá suporte à execução de aplicações Bag-Of-Tasks (BoT) - aplicações paralelas cujas tarefas são independentes [Cirne et al. 2003, Smith and Shrivastava 1996]. Os principais componentes do OurGrid são o *broker*, o *worker*, o *peer* e o *discovery service*. *Brokers* e *workers* fazem parte de um *site* administrado por um *peer*. *Peers* se descobrem por meio do *discovery service*, formando uma comunidade OurGrid. A arquitetura do OurGrid é ilustrada na Figura 1.

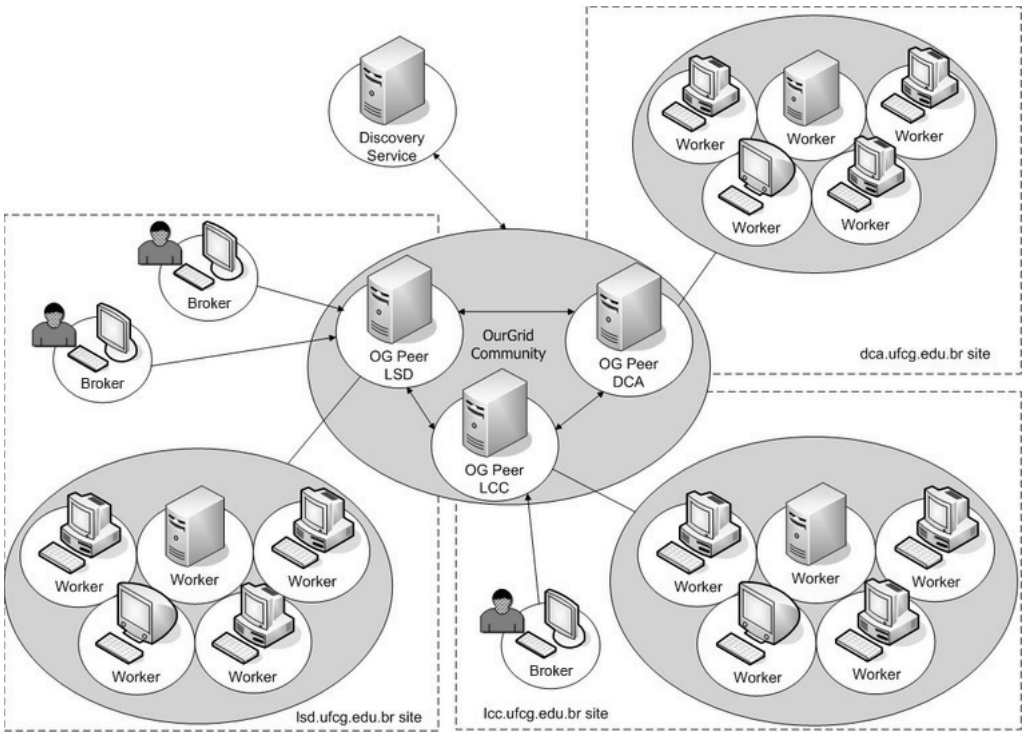

Figura 1. Arquitetura do OurGrid

O *broker* é a entidade que interage com o usuário e é responsável pela submissão dos *jobs*, o *worker* roda nos recursos computacionais destinados à execução de tarefas e o *peer* é responsável pela descoberta de recursos.

Atualmente o OurGrid faz uso do *framework* de comunicação Commune [Vilar 2010], que implementa um detector de falhas de resposta binária e com tempo de detecção estático. Durante uma execução de tarefa, se um *worker* é suspeito de falha, seja por indisponibilidade da máquina ou preempção, o *broker* marca essa execução como falha.

Utilizar o OurGrid como caso de uso é interessante pois o *middleware* tem aspectos arquiteturais e de implantação que dão margem a melhorias de desempenho com o uso de técnicas de detecção avançadas. Por exemplo, detectores adaptativos são conhecidos por funcionarem bem em ambientes de execução com alta flutuação de latência, como WAN e Internet, que são, tipicamente, os ambientes de execução do OurGrid.

Nesse trabalho serão analisadas as falhas de *workers* em execucões abortadas ou preemptadas. Para isso será utilizado o OurSim, simulador do OurGrid orientado a eventos. O OurSim não separa o tempo de detecção de falha dos componentes do tempo total de computação das tarefas. Dessa forma, será necessário modificá-lo para permitir essa separação, o que habilitará a implantação de um módulo detector de falhas e dos próprios detectores adaptativos.

4. Solução e experimentação

4.1. Mecanismos escolhidos

Dos mecanismos de detecção estudados até o momento, aqueles propostos por Chen et al [Chen et al. 2000], Bertier et al [Bertier et al. 2002] e Hayashibara et al. [Hayashibara et al. 2004] têm se destacado na literatura como base de comparação e avaliação. Nesse sentido, esses serão os mecanismos de detecção a serem implementados no OurSim e avaliados nesse trabalho.

4.2. Adaptação do OurSim

O OurSim é um simulador do OurGrid orientado a eventos. Ele é capaz de tratar recursos dedicados, voláteis, heterogêneos e multiprocessados, tem sua arquitetura baseada no padrão de projeto *Observer* e permite enfileiramento de eventos de tarefas e disponibilidade por demanda.

As mudanças de estado de *worker* no OurSim são executadas na ocorrência dos eventos *workerAvailable* e *workerUnavailable*, que, por sua vez, são criados por um gerador sintético de disponibilidade, baseado em dados históricos reais de disponibilidade de *workers* da comunidade OurGrid.

Apesar de modelar a disponibilidade, o OurSim não leva em consideração os atrasos relacionados a falhas de *worker*. Para isso, foi desenvolvido um módulo detector de falhas, integrado por composição na entidade *Peer*. A estratégia de detecção pode então ser configurada via linha de comando, como um dos parâmetros do detector.

Com o módulo de detecção implementado, foi necessário implementar um módulo gerador de *heartbeats* sintéticos, que é responsável por gerar os atrasos dos *heartbeats*, dados uma distribuição e um atraso médio. Esses valores de chegada de *heartbeats* são necessários para alimentar o módulo de detecção de falhas.

Além dos eventos *workerAvailable* e *workerUnavailable*, utilizou-se mais dois eventos de disponibilidade, o *workerUp* e o *workerDown*. Dessa forma, quando um

evento de *workerDown* é disparado, um evento de *workerUnavailable* é escalonado para o tempo do evento atual somado do *timeout* fornecido pelo módulo detector de falhas.

O diagrama de sequência da Figura 2 ilustra o funcionamento do OurSim com as modificações supracitadas.

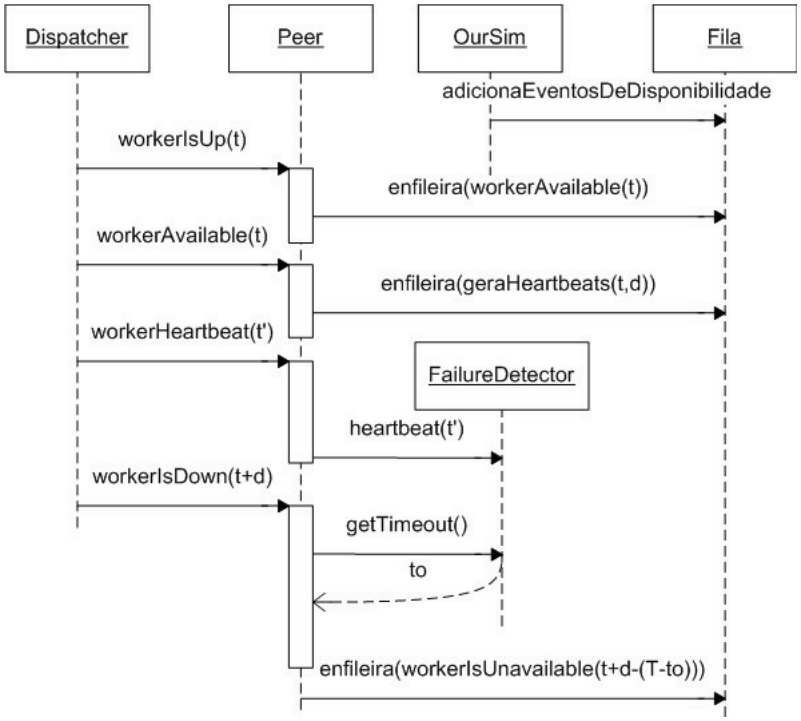

Figura 2. Funcionamento do OurSim

4.3. Hipóteses

Nesse trabalho será analisado o impacto da utilização de três detectores de falha adaptativos no OurGrid por meio de simulação. O patamar de comparação sempre será o detector de *timeout* estático, que é o mecanismo atualmente implementado no *middleware* em questão. Dessa forma podemos definir três pares de hipóteses nula e alternativa a serem testadas:

H'_0: Não há diminuição do *makespan* das tarefas ao utilizar o mecanismo proposto por Chen et al.

H'_1: Há diminuição do *makespan* das tarefas ao utilizar o mecanismo proposto por Chen et al.

H''_0: Não há diminuição do *makespan* das tarefas ao utilizar o mecanismo proposto por Bertier et al.

H''_1: Há diminuição do *makespan* das tarefas ao utilizar o mecanismo proposto por Bertier et al.

H'''_0: Não há diminuição do *makespan* das tarefas ao utilizar o mecanismo proposto por Hayashibara et al.

H'''_1: Há diminuição do *makespan* das tarefas ao utilizar o mecanismo proposto por Hayashibara et al.

4.4. Planejamento experimental

O *workload* utilizado nos experimentos foi gerado a partir de traços reais de submissão de tarefas em grades P2P, utilizando o modelo de geração de carga sintética descrito em [Carvalho 2010], que descreve o comportamento dos usuários da grade pelo intervalo entre submissões, e as características das aplicações pelo tempo de execução de tarefas e tempo de execução de *jobs*.

O *workload* gerado para esse experimento representa a execução de *jobs bag-of-task* durante uma semana, em uma comunidade de 50 *peers*, cada *site* com 50 *workers*, 210 usuários submetendo 1860 *jobs* com tamanho médio de 248 *tasks* e desvio padrão de 269 *tasks*. Cada *task* dura em média 1949,03 s, com desvio padrão de 1677,67 s. Foi considerado um escalonamento de tarefas com replicação 3 [OurGrid 2010], valor padrão no OurGrid.

Os *heartbeats* foram gerados com base em um atraso médio de 500ms, emulando um *link* intercontinental e de alta latência, e o intervalo de interrogação (*ping*) dos detectores utilizado nos experimentos foi definido de acordo com o valor padrão do OurGrid, que é de 60 segundos [OurGrid 2010].

Quatro cenários foram executados, um para cada mecanismo de detecção de falha. O mecanismo de *timeout* foi configurado com tempo de detecção de 300 segundos, que é o valor padrão do OurGrid; o Phi-Accrual foi configurado com *threshold* igual a 0.5; o mecanismo proposto por Chen foi configurado com α=5s; e o detector proposto por Bertier foi configurado com β=1, γ=0.1, ϕ=4 e passo de moderação de 5s. Esses valores foram escolhidos com base nos experimentos descritos em [Bertier et al. 2002, Hayashibara et al. 2004, Barros 2010] e nas condições dos cenários desta simulação.

Nesse trabalho apenas as falhas de *workers* foram consideradas no cálculo do *makespan* das tarefas.

O experimento foi definido com nível de replicação cinco, ou seja, cada cenário foi executado cinco vezes.

5. Resultados e validação

Dado que se pretende verificar se existe diferença no *makespan* de tarefas entre as abordagens adaptativas e o detector de *timeout* estático, foi utilizado um teste-t pareado, com o valor de 0.95 de coeficiente de confiança, para realizar a comparação entre todas as tarefas para todas as replicações.

Com base no teste-t foi possível refutar todas as hipóteses nulas, ou seja, o uso dos detectores adaptativos provocou uma diminuição estatisticamente significativa no *makespan* das tarefas.

Na Figura 3 é possível perceber a evolução da diminuição de *makespan* das tarefas ao longo de tempo comparado ao *makespan* obtido pelo detector de *timeout* estático, fato decorrente da adaptação do tempo de detecção, característica dos detectores adaptativos. Nota-se que os detectores propostos por Chen e Hayashibara (Phi-Accrual) são os que obtém melhor desempenho no geral - fato também notado por Hayashibara na seção de avaliação do seu trabalho de comparação [Hayashibara et al. 2004].

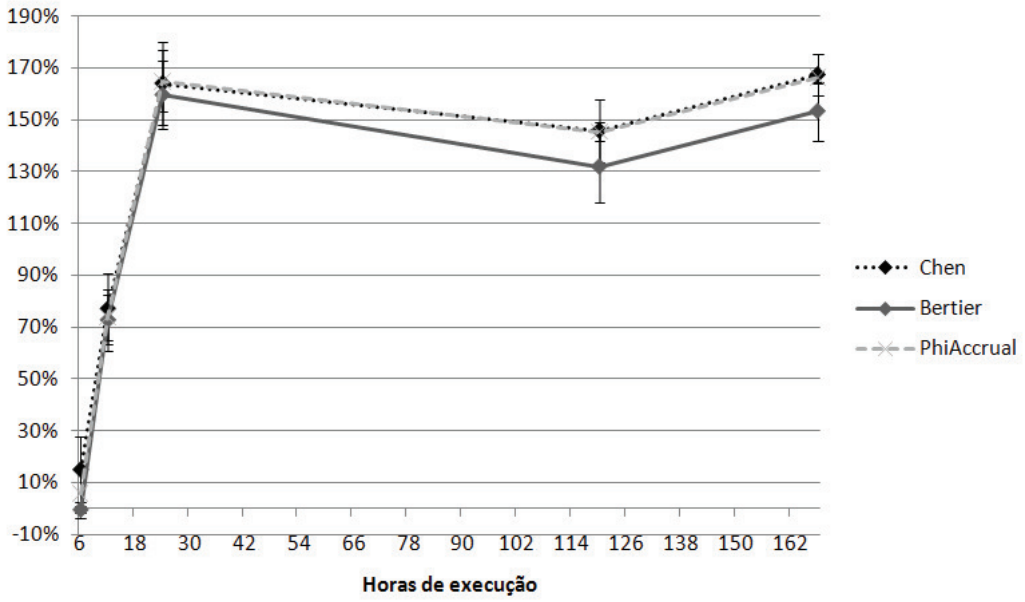

Figura 3. Diminuição média do makespan das tarefas em relação ao detector de timeout estático

A diminuição do *makespan* das tarefas reflete num menor desperdício de recursos, porém, apesar da notável diminuição nos valores dessa métrica, a métrica *makespan* de *job* apresentou pouca variação (entre 2% e 5%) quando comparada com o detector estático. Isso acontece porque o *makespan* de *jobs* está intimamente relacionado ao tempo de execução da réplica de maior duração que foi finalizada com sucesso. Analisando essa métrica, especialmente, não foi possível notar um efeito substancial do mecanismo adaptativo de detecção de falhas.

6. Conclusões

Nesse trabalho foi feita uma análise preliminar do impacto do uso de detectores de falhas sofisticados em sistemas distribuídos, tratando o *middleware* de grades computacionais OurGrid como caso de uso. Por meio de simulação foi comprovado o efeito dos mecanismos de detecção de falha propostos por Chen, Bertier e Hayashibara no *makespan* das tarefas submetidas à grade. Além disso, constatou-se que, considerando apenas as falhas de *worker*, não há um ganho significativo no *makespan* de *jobs*, uma vez que essa métrica é muito mais consistente que o *makespan* de tarefa.

Esse trabalho serve como motivação para futuros trabalhos com o objetivo de aproximar o estado da arte ao estado da prática em detectores de falha. Como discutido anteriormente, é perceptível que o uso de detectores de falha sofisticados impacta positivamente em métricas orientadas à aplicação, como o tempo de computação, mapeado para *makespan* nesse caso do uso. Porém deve existir um esforço para esclarecer os reais benefícios dessa utilização em diferentes cenários para diferentes sistemas.

Há uma série de frentes para as quais é possível estender o trabalho atual. Inicialmente é necessário considerar falsas suspeições no ciclo de vida dos objetos monitorados, pois as métricas de tempo para detectores não fazem sentido quando não acompanhadas da contagem de falsas suspeições.

Depois, implementar os detectores de falha no próprio OurGrid é fundamental, uma vez que o *middleware* tem aspectos arquiteturais e de implantação que dão margem a melhorias de desempenho com o uso de técnicas de detecção avançadas. Por exemplo, detectores adaptativos são conhecidos por funcionarem bem em ambientes de execução com alta flutuação de latência, como WAN e Internet, que são, tipicamente, os ambientes de execução do OurGrid. Além disso, no OurGrid diferentes níveis de confiabilidade podem ser aplicados em diferentes momentos (diferentes estados do *worker*), ou mesmo quando diferentes entidades são monitoradas (*workers* remotos e locais), o que evidencia os benefícios da utilização do paradigma *accrual* de detecção de falhas, se tornando um excelente caso de uso para a área dessa pesquisa, em suas diferentes frentes.

Durante a experimentação com o OurGrid será possível considerar falhas entre outros componentes além de *workers*, o que poderá afetar diretamente no *makespan* de *jobs*, métrica não exercitada nesse trabalho.

Além disso, outras métricas relacionadas à aplicação deverão ser estudadas, como, por exemplo, a quantidade de recursos que é desperdiçada ao longo do tempo da computação. No OurGrid essa métrica é mapeada para *badput*, que é o tempo de execução das réplicas preemptadas ou abortadas.

Referências

Barros, A. (2010). Zookeeper failure detector model. http://wiki.apache.org/hadoop/ZooKeeper/GSoCFailureDetector.

Bertier, M., Marin, O., and Sens, P. (2002). Implementation and performance evaluation of an adaptable failure detector. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 354 – 363.

Carvalho, M. W. A. (2010). Predição da Qualidade de Serviço em Grades Computacionais P2P. Master's thesis, Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brasil.

Chandra, T. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

Chandra, T. D., Hadzilacos, V., and Toueg, S. (1996). The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722.

Chen, W., Toueg, S., and Aguilera, M. K. (2000). On the quality of service of failure detectors. In *International Conference on Dependable Systems and Networks (DSN'2000)*, pages 191–200, New York, USA. IEEE Computer Society.

Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., and Mowbray, M. (2006). Labs of the World, Unite!!! *Journal of Grid Computing*, 4(3):225–246.

Cirne, W., Brasileiro, F., Sauvé, J., Andrade, N., Paranhos, D., Santos-neto, E., Medeiros, R., and Gr, F. C. (2003). Grid computing for bag of tasks applications. In *In Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment*.

Défago, X., Hayashibara, N., Katayama, T., and Katayama, N. H. T. (2003). On the design of a failure detection service for large-scale distributed systems. In *In Proc. Intl. Symposium Towards Peta-Bit Ultra-Networks (PBit 2003*, pages 88–95.

Défago, X., Urbán, P., Hayashibara, N., and Katayama, T. (2005). Definition and specification of accrual failure detectors. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'2005)*, pages 206–215, Yokohama, Japan. IEEE Computer Society.

Falai, L. and Bondavalli, A. (2005). Experimental evaluation of the qos of failure detectors on wide area network. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, DSN '05, pages 624–633, Washington, DC, USA. IEEE Computer Society.

Felber, P., Défago, X., Guerraoui, R., and Oser, P. (1999). Failure detectors as first class objects. *Distributed Objects and Applications, International Symposium on*, 0:132.

Fischer, M. J., Lynch, N. A., and Paterson, M. D. (1985). Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374–382.

Fraga, E. (2010). Oursim - an opportunistic peer-to-peer desktop grid simulator. http://redmine.lsd.ufcg.edu.br/wiki/oursim.

Gupta, I., Chandra, T., and Goldszmidt, G. (2001). On scalable and efficient distributed failure detectors. In *Proceedings of the* 20^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC'2001)*, pages 170–179, Newport, Rhode Island.

Hayashibara, N., Défago, X., Yared, R., and Katayama, T. (2004). The φ accrual failure detector. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, pages 66–78, Florianópolis, Brazil. IEEE Computer Society.

Junqueira, F. P. and Reed, B. C. (2009). The life and times of a zookeeper. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, SPAA '09, pages 46–46, New York, NY, USA. ACM.

Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44:35–40.

Larrea, M., Fernandez, A., and Arévalo, S. (2000). Optimal implementation of the weakest failure detector for solving consensus. In *19th IEEE Symposium on Reliable Distributed Systems (SRDS'2000)*, pages 52–59, Nurnberg, Germany. IEEE Computer Society.

Lung, L. C., Favarim, F., Santos, G. T., and Correia, M. (2006). An infrastructure for adaptive fault tolerance on ft-corba. In *Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, ISORC '06, pages 504–511, Washington, DC, USA. IEEE Computer Society.

Nunes, R. C. and Jansch-Pôrto, I. (2002). Modelling communication delays in distributed systems using time series. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'2002)*, pages 268–273, Osaka, Japan. IEEE Computer Society.

OurGrid (2010). Ourgrid middleware - development page. http://redmine.lsd.ufcg.edu.br/projects/show/ourgrid.

Paxson, V. (2000). Computing tcp's retransmission timer. http://www.faqs.org/rfcs/rfc2988.html.

Smith, J. and Shrivastava, S. (1996). A system for fault-tolerant execution of data and compute intensive programs over a network of workstations.

Su, W.-C., Wang, S.-C., and Kuo, S.-Y. (2001). Failure detection mechanism for distributed object computing using corba. *Pacific Rim International Symposium on Dependable Computing, IEEE*, 0:273.

Vilar, R. (2010). Commune - a communication platform for asynchronous distributed objects. http://redmine.lsd.ufcg.edu.br/projects/commune/.

Xiong, N. and Defago, X. (2007). Ed fd: Improving the phi accrual failure detector. *Research report*, 2007:1–29.

Zhuang, S. Q., Geels, D., Stoica, I., and Katz, R. H. (2003). On failure detection algorithms in overlay networks. In *IN IEEE INFOCOM*.