Uma Nova Abordagem Para Otimizar a Comunicação Entre Detectores de Defeitos

Rogério Turchetti
Centro Universitário Franciscano
Núcleo de Ext. e Pesq. em Informática – NEPI
turchetti@unifra.br

Raul Ceretta Nunes
Universidade Federal de Santa Maria
Dep. de Eletrônica e Computação – GMICRO
ceretta@inf.ufsm.br

Resumo

Detectores de defeitos (FDs) não confiáveis são utilizados como bloco básico na especificação e implementação de tolerância a falhas em sistemas distribuídos assíncronos. Um exemplo típico de sistemas distribuídos assíncronos e de larga escala é a Internet. Neste contexto, FDs tradicionais apresentam problemas, uma vez que seu projeto destina-se à redes controladas (LAN). Um problema a ser tratado é a explosão de mensagens, pois em sistemas de larga escala, onde o número de processos e os atrasos são imprevisíveis o problema da explosão de mensagens pode comprometer o desempenho do serviço de detecção de defeitos e a escalabilidade da aplicação. Neste sentido, este artigo trata do problema da explosão de mensagens propondo uma abordagem genérica e prática que utiliza o reaproveitamento de mensagens para suprir mensagens de controle nos FDs.

1. Introdução

Um sistema tolerante a falhas parte do princípio da replicação de recursos e/ou componentes [8]. Entretanto, com a replicação destes surgem outros problemas que devem ser tratados, como por exemplo, fazer com que eventos sejam executados na mesma seqüência em todas as réplicas do sistema, a fim de manter a consistência dos dados. Freqüentemente a coordenação consistente de tais eventos exige alguma forma de acordo distribuído, mas a realização de tal tarefa não é possível em sistemas assíncronos sujeito a falhas. Isso se deve a impossibilidade em determinar quando um processo está falho ou simplesmente mais lento que os demais [7]. Frente a essa limitação, detectores de defeitos (*FDs*) surgem como uma solução [3].

Módulos de detecção de defeitos trabalham como um oráculo encapsulando o problema do indeterminismo, isto é, eles tentam descobrir os estados funcionais dos processos e fornecem informações suficientes para permitir soluções determinísticas. Em síntese, *FDs* trabalham em função da formação e da manutenção de uma visão que consiste nos processos suspeitos. Numa abordagem tradicional, o monitoramento de defeitos é executado entre todos os processos, ou seja, cada processo executa o monitoramento trocando informações diretamente com os demais processos. Num ambiente controlado, como numa rede local ou *cluster*, a explosão de mensagens pode não ser um problema muito comum, pois o atraso é pequeno e a largura da banda (vazão) é grande, se comparados a uma rede de larga escala como a Internet. Em sistemas distribuídos de larga escala, como em alguns grids, onde o número de processos e os atrasos são imprevisíveis e a largura da banda é restrita, o problema da explosão de mensagens gerado pelo grande número de mensagens de controle, pode comprometer o serviço de detecção de defeitos e a escalabilidade do sistema.

Neste sentido este trabalho explora o problema da explosão de mensagens causadas pelas ações de monitoramento dos *FDs*. De acordo com [9] este é um dos 6 problemas que causam transtorno na aplicabilidade de *FDs* tradicionais em sistemas de larga escala. Neste sentido, diversas soluções ao referido problema foram propostas. Sargent et al. [13], para obter algoritmos eficiente em termos de número de mensagens propuseram a especialização dos *FDs* junto a protocolos de consenso (aplicação), mas os algoritmos dificilmente podem ser aplicados à outras aplicações. Para economizar mensagens, de maneira similar também foram definidos *FDs* que operam segundo topologias lógicas de rede, por exemplo, topologia em anel [10], em estrela [11] e hierárquica [5, 1, 2]. Preocupados com o custo (perda de desempenho) quando *FDs* são utilizados, Fetzer et al. [6] propuseram o reaproveitamento de mensagens da aplicação para suprir mensagens dos *FDs*.

Neste artigo, é proposta uma abordagem para redução de mensagens de controle *FDs*, a qual difere das demais por: (i) reaproveita mensagens da aplicação e do próprio *FD*; e (ii) pode ser combinada a diversos algoritmos de monitora-

mento. O ponto chave da abordagem proposta é a alteração semântica das mensagens do *FD*, o que possibilita suprimir mensagens de controle e ser aplicada a qualquer algoritmo de monitoramento. O artigo analisa tanto o impacto da abordagem na redução de mensagens de controle como também sua influência na Qualidade de Serviço (QoS) dos *FDs*. Um serviço de detecção de defeitos adaptativo (AFDService) executando sobre um ambiente de larga escala (PlanetLab [12]) formaram o ambiente de experimentação.

Este artigo está organizado como segue. A seção 2 apresenta o modelo de sistema e algumas definições. Na seção 3 é detalhada a abordagem proposta, os experimentos são mostrados na seção 4. Por fim, na seção 5 apresenta-se as considerações finais.

2. Modelo de Sistema

Considera-se um sistema distribuído composto por um número finito de processos $\Omega = \{p_1, p_2, ..., p_n\}$ onde cada processo pode se comunicar com qualquer outro processo do sistema. Para todo processo $p_i \in \Omega$ há um relógio interno que funciona independente dos demais relógios. Além disto, assume-se neste trabalho o modelo parcialmente síncrono proposto por [3]. Tal modelo considera que para toda execução ou comunicação existem limites temporais, entretanto esses limites não são conhecidos antes de um tempo *St* (*Stabilization time*) desconhecido. O limite de temporização torna-se conhecido após o sistema atingir *St*.

Assume-se que os processos somente suportam falhas por colapso *(crash)*, ou seja, por suspender sua execução prematuramente. A comunicação entre processos é realizada pelo envio e recebimento de mensagens, através de um canal de comunicação confiável (*reliable channel*), ou seja, o canal não **cria**, **altera**, **duplica** e nem **perde** mensagens de controle. E ainda, o canal não requer ser FIFO (*First-In First-Out*).

3 A Nova Abordagem

A ação de monitorar processos em sistemas distribuídos é baseado na troca de mensagens de controle [5]. Dependendo do algoritmo utilizado, do número de processos participantes ou da característica do ambiente, esta ação poderá ocasionar a sobrecarga dos canais de comunicação, devido a explosão de mensagens gerada. Neste sentido, para reduzir a carga da rede, a presente seção explora o uso de uma nova abordagem que atrasa o envio das mensagens de controle sempre que possível.

3.1 Adaptação da Taxa de Freqüência (ATF)

A estratégia ATF consiste no reaproveitamento de mensagens de controle reutilizando mensagens dos próprios algoritmos de detecção de defeitos. A ATF altera a semântica das mensagens de detecção, ou seja, o significado das mensagens de controle de um detector que requisita estados aos processos monitorados.

Tradicionalmente estes *FDs* monitoram os processos enviando, a cada Δ_i unidades de tempo mensagens de requisição de estado `AreYouAlive` para um processo monitorado e aguarda por uma resposta (`IAmAlive`). Para garantir que o processo monitorado esteja operacional é necessário que a resposta seja recebida dentro de um certo limite de tempo. Entretanto, aplicando a estratégia ATF, um processo monitor assume que qualquer mensagem recebida de um processo monitorado (`AreYouAlive` ou `IAmAlive`) indica a vivacidade/acessibilidade do processo emissor naquele instante. Além disto, ele atrasa o envio da requisição de estado reinicializando o temporizador do Δ_i. Ressalta-se que, embora a nova abordagem seja designada para um propósito geral, a estratégia ATF aplica-se a todos os algoritmos de detecção de defeitos que realizam requisições de estados aos processos monitorados.

Para demonstrar o funcionamento da estratégia ATF, assuma o seguinte cenário: um processo p_i monitora, e é monitorado por outro processo p_j. A cada Δ_i p_i e p_j deveriam enviar mensagens `AreYouAlive` para verificar o estado do processo vizinho. Entretanto, a cada instante que um dos processos receber uma mensagem `AreYouAlive`, o detector assume que o processo emissor está operacional e reinicializa o relógio que controla Δ_i. Como resultado, um processo monitor somente enviará uma mensagem `AreYouAlive` se e somente se ele não receber nenhuma mensagem de controle do processo monitorado em Δ_i instantes de tempo.

3.1.1 Algoritmo Para a Estratégia ATF

O algoritmo para a estratégia ATF é apresentado na figura 1, ele está dividido em duas etapas (*Task 1* e *Task 2*) distintas, onde processos monitores executam as tarefas *Task 1* e *Task 2* e processos monitorados executam somente a *Task 2*. Para facilitar a especificação do algoritmo, será utilizado como exemplo ilustrativo dois processos p_i e p_j onde p_j é monitorado pelo processo p_i.

```
1: Todo processo p_i executa:
2:
3: seqNumber ← 0
4: ∀ p_j ∈ Ω : Δ_i^{p_j} ← default frequency
5:     Δ_to^{p_j} ← default timeout
6: L^g_{p_i} ← ∅ and L^l_{p_i} ← ∅                {lista global e lista local respectivamente}
7: received ← true
8: cobegin
9: Task 1:
10:    loop
11:       if Δ_i^{p_j} = 0 then
12:          send (AreYouAlive, L^g_{p_i}, seqNumber) to p_j
13:          seqNumber ← seqNumber + 1
14:          restart Δ_to^{p_j} and Δ_i^{p_j}
15:          received ← false
16:       end if
17:       if Δ_to^{p_j} = 0 then
18:          if not received then
19:             L^l_{p_i} ← L^l_{p_i} ⋃ {p_j}
20:             L^g_{p_i} ← L^g_{p_i} ⋃ {p_j}
21:          end if
22:       end if
23:    end loop
24: Task 2:
25:    forever
26:    upon
27:       received message m from a process p_j
28:       at ← arrivalTime
29:       case m = AreYouAlive or m = IAmAlive
30:          if m = AreYouAlive then send (IAmAlive) to p_j
31:             Δ_i^{p_j} ← at + defaultfrequency      {atualiza o próximo envio}
32:             if p_j ∈ L^l_{p_i} then
33:                L^l_{p_i} ← L^l_{p_i} - {p_j}
34:                Δ_to^{p_j} ← Δ_to^{p_j} + 1
35:             end if
36:             L^g_{p_i} ← L^g_{p_j} ⋃ L^l_{p_i} - {p_i, p_j}
37:             received ← true
38:          end if
39:          if m = IAmAlive then
40:             if p_j ∈ L^l_{p_i} then
41:                L^l_{p_i} ← L^l_{p_i} - {p_j}
42:                L^g_{p_i} ← L^g_{p_i} - {p_j}
43:                Δ_to^{p_j} ← Δ_to^{p_j} + 1
44:             end if
45:             received ← true
46:          end if
47:       end case
48:    end forever
49: coend
```

Figura 1. Algoritmo com a estratégia ATF

Inicialmente, todo processo p_i executa algumas inicializações. Considerando que $L^l_{p_i}$ representa a lista local de processos suspeitos de p_i e que Ω representa o conjunto dos processos participantes do sistema (vide seção 2), o algoritmo assegura que nenhum processo é suspeito pelos demais processos do sistema, $\forall\ p_i : L^l_{p_i} = \emptyset$, e que todo processo inicia sua execução monitorando os demais processos, garantindo que $\forall\ p_i \in \Omega$ há um módulo de detecção vinculado. Após inicializado, cada processo cumpre suas tarefas como descrito a seguir (figura 1):

Na *Task 1* (linhas 9-23) o processo p_i dispara as mensagens de monitoramento quando a periodicidade (Δ_i) atingir o valor de 0 (linha 11) aos seus alvos. No método *send* (linha 12) são passados por parâmetro três valores: o tipo da mensagem, o número de seqüência da mensagem e sua lista global indicando os processos suspeitos. A lista global permite obter uma visão consistente entre diferentes detectores de defeitos. Se nenhuma mensagem for recebida num período Δ_{to}, p_i inicia a suspeitar de p_j adicionando-o em sua lista de processos suspeitos (linhas 19-20).

Na *Task 2* (linhas 24-48) as mensagens (*m*) são recebidas pelos processos (linha 27). Caso m seja do tipo `AreYouAlive` (linha 30) p_i responde ao processo com uma mensagem `IAmAlive` e assume que p_j está operacional reajustando o temporizador $\Delta_i^{p_j}$ (linha 31). Esta estratégia permite reduzir o número de mensagens enviadas fazendo a seguinte analogia: uma mensagem `AreYouAlive` é equivalente a uma mensagem `IAmAlive`. Assim p_i verifica em sua lista local se p_j estava como suspeito, se sim ele retira o processo de sua lista de suspeitos local (linha 33) e incrementa o valor de $\Delta_{to}^{p_j}$ (linha 34). O incremento do *timeout* significa que o algoritmo cometeu um engano por ainda não ter atingido St. O processo p_i extrai da mensagem `AreYouAlive` recebida, a lista global de processos suspeitos realizando uma fusão com sua lista global (linha 36). Caso m seja do tipo `IAmAlive` (linha 39) p_i verifica em sua lista local se p_j estava como suspeito, se sim ele retira o processo de sua lista de suspeitos local e global (linhas 41 e 42) e incrementa o valor de $\Delta_{to}^{p_j}$(linha 43). Por fim, sempre que p_i indicar a operacionalidade do processo p_j no atual instante, ele passa a variável *received* a receber o valor de *true* (linhas 37 e 45).

Este algoritmo assegura as propriedades necessárias para a implementação de um *FD* da classe $\Diamond\mathcal{P}$[1].

3.2 Aproveitamento de Mensagens da Aplicação (AMA)

A estratégia AMA segue a mesma filosofia da estratégia ATF, no que diz respeito ao atraso das mensagens de controle. No entanto, a estratégia AMA reaproveita mensagens geradas pelas aplicações para suprir mensagens de controle. Neste caso, a semântica das mensagens de controle também são alteradas, uma vez que, qualquer mensagem da aplicação recebida de um processo monitorado indica a vivacidade do processo emissor.

[1] Os algoritmos garantem a classe $\Diamond\mathcal{P}$ por assegurarem as propriedades: **Precisão Eventualmente Forte**: esta propriedade foi cumprida por aumentar o *timeout* a cada suspeita errada; e **Abrangência Forte**: foi garantida através da implementação de uma lista global de processos suspeitos.

A solução proposta assume um serviço de detecção de defeitos trabalhando entre a aplicação cliente e o *kernel* do sistema operacional. Tendo em vista que um *FD* monitora processos através de duas primitivas básicas: *send* e *receive*, e que em sistemas distribuídos as aplicações trocam mensagens freqüentemente, pode-se oferecer as aplicações clientes uma interface que permite-as utilizar as primitivas oferecidas pela camada de detecção. Ciente do fluxo de mensagens da aplicação, o *FD* pode utilizá-lo para suprir mensagens de controle. Como resultado o *FD* somente irá gerar mensagens caso as aplicações não estiverem trocando informações. A estratégia de aproveitar mensagens da aplicação para reduzir mensagens de controle é amplamente utilizada, mas a proposta neste trabalho possui algumas inovações, como por exemplo, combiná-la com a filosofia da estratégia ATF. Neste contexto, um algoritmo que implementa ATF+AMA assume que qualquer mensagem do tipo `AreYouAlive`, `IAmAlive` ou `ApplicationMessage` recebida de um processo monitorado indica a vivacidade do processo emissor naquele instante.

A estratégia de reaproveitamento de mensagens foi muito bem explorada no algoritmo de detecção denominado FD preguiçoso (*Lazy*) [6]. Nele, quando o FD associado a uma instância *q* da aplicação recebe uma mensagem proveniente de uma instância *p* ele deve retornar uma mensagem de confirmação (*ack*), pois o protocolo depende de uma confirmação. Na estratégia AMA o *ack* não é necessário, pois as mensagens da aplicação são utilizadas como mensagens `IAmAlive`, sendo contabilizadas somente no processo que recebe a mensagem da aplicação (`ApplicationMessage`). Como resultado tem-se um serviço de monitoramento com menor custo.

Um aspecto importante da estratégia AMA é que as mensagens da aplicação são aproveitadas para detectar a operacionalidade dos processos e não defeitos, logo, não existem *timeouts* para as mensagens da aplicação. Além disto, nenhuma mensagem a mais é necessária para cada mensagem da aplicação enviada, e segundo aspecto é que esta estratégia pode ser extendida a diversos algoritmos de detecção.

3.2.1 Algoritmo Para a Estratégia AMA

O algoritmo para AMA segue a mesma especificação feita na seção 3.1.1. Após a inicialização do algoritmo, cada processo cumpre suas tarefas como descrito a seguir (figura 2): na *Task 2* os processos trabalham da mesma forma que foi especificado na *Task 2* da estratégia ATF, exceto pelo fato de que esta estratégia também assume mensagens da aplicação. As linhas 46 à 55 demonstram como é tratada uma mensagem recebida de uma aplicação localizada no processo p_j.

Tão logo o processo p_i recebe uma mensagem do tipo `ApplicationMessage`, ele assume que p_j está operacional e por ter recebido uma mensagem da aplicação ele reajusta o temporizador $\Delta_i^{p_j}$ (linha 47) atrasando a próxima mensagem. Logo depois, p_i verifica o estado do processo emissor na lista local de processos suspeitos (linha: 48), caso o processo emissor constar na lista de processos suspeitos ele será retirado da lista local e global, pois uma falsa suspeita haverá sido levantada. Devido a isto o *FD* incrementa o valor de $\Delta_{to}^{p_j}$ (linha 51) em busca de atingir St para não ocorrer mais falhas de temporização. Por se tratar de uma mensagem da aplicação cliente o *FD* realiza um *deliver* (linha 54) para repassar e entregar a mensagem destinada a camada superior de aplicação.

```
.
.                                                  {idem ao algoritmo da figura 1}
.
24: Task 2:
25:     forever
26:     upon
27:         received message m from a process p_j
28:         at ← arrivalTime
.
.                                                  {idem ao algoritmo da figura 1}
.
46:           if m = ApplicationMessage then
47:             Δ_i^{p_j} ← at + defaultfrequency      {atualiza o próximo envio}
48:             if p_j ∈ L_{p_i}^l then
49:                 L_{p_i}^l ← L_{p_i}^l - {p_j}
50:                 L_{p_i}^g ← L_{p_i}^g - {p_j}
51:                 Δ_{to}^{p_j} ← Δ_{to}^{p_j} + 1
52:             end if
53:             received ← true
54:             deliver (m)
55:           end if
56:         end case
57:     end forever
58: coend
```

Figura 2. Algoritmo com a estratégia AMA

4 Ambiente de Execução e Experimentos

Esta seção apresenta os experimentos e detalhes sobre a realização de testes aplicados no presente trabalho. Inicialmente avalia-se o impacto de redução do número de mensagens de controle. Após será avaliado o impacto na QoS dos algoritmos propostos. Uma breve descrição sobre o cenário utilizado para a execução dos testes será apresentada a seguir.

4.1 Ambiente de Execução

Avaliar e comparar serviços não é uma tarefa trivial, principalmente em ambientes onde os componentes estão

dispostos distribuidamente. A dificuldade não está somente em determinar quais métricas devem ser utilizadas e sim como aplicá-las. Montar um ambiente distribuído envolve ter disponível uma gama de dispositivos, o que muitas vezes torna-se inviável. Para o presente trabalho optou-se pela utilização de um ambiente real e de grande escala. A solução encontrada para a execução dos experimentos foi o uso do laboratório mundial denominado PlanetLab [12]. A figura 3 apresenta os nodos utilizados, bem como a distribuição deles no laboratório virtual, composto por 8 processos, um por nodo, todos executando no mesmo sistema operacional e mesmo $hardware$.

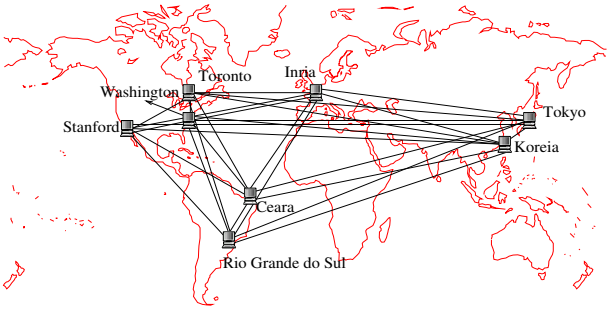
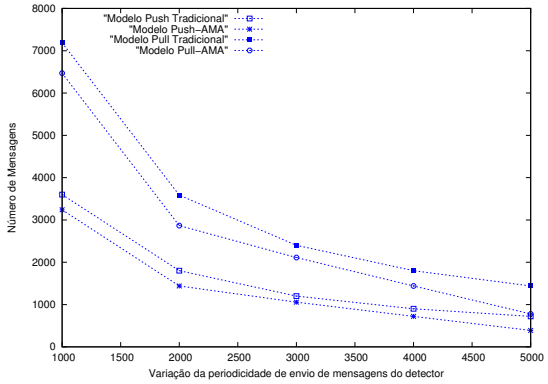
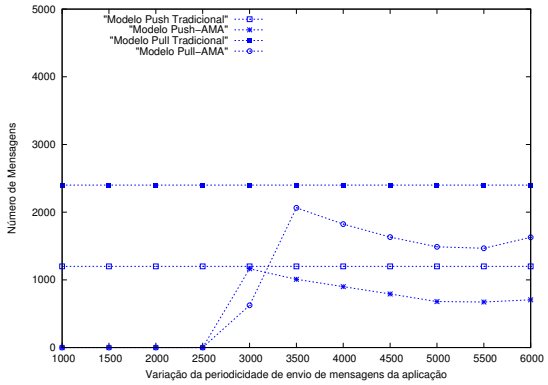


Figura 3. Distribuição de processos no Planet-Lab



(a) Variando o Δ_i



(b) Variando a periodicidade da aplicação cliente

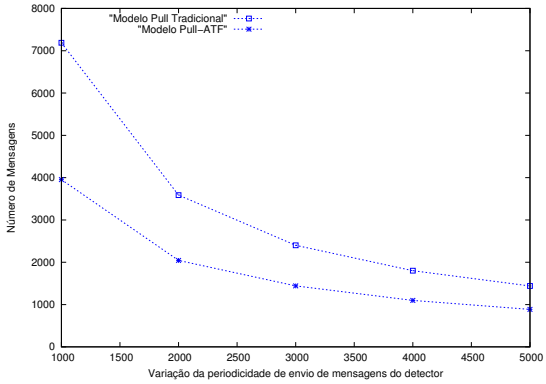
Figura 4. Experimentos para a estratégia AMA

4.2 Experimentos

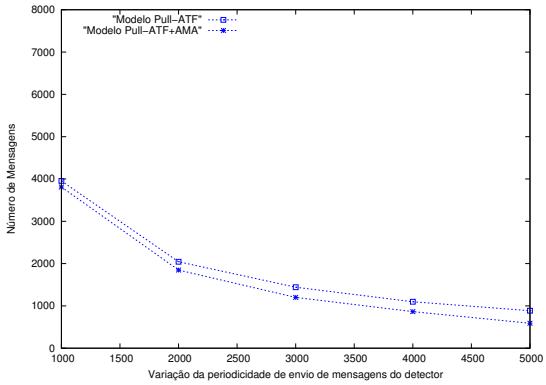
4.2.1 Número de Mensagens de Controle Enviadas

Nesta seção são realizados os experimentos relacionados ao número de mensagens enviadas no canal de comunicação em detrimento das ações de monitoramento dos algoritmos de detecção de defeitos. Para as primeiras análises variou-se apenas a periodicidade de envio das mensagens. O objetivo foi verificar se as estratégias propostas obtêm melhores resultados em períodos com maior ou menor carga na rede. Os intervalos de amostras foram em períodos de 60 minutos, sendo que a amostragem foi realizada em diferentes horários no decorrer do dia. Os valores que serão apresentados correspondem a dados médios. Os experimentos com a estratégia AMA, utilizou-se uma aplicação sintética que troca informações em períodos constantes de 10000 ms (gráfico 4(a)).

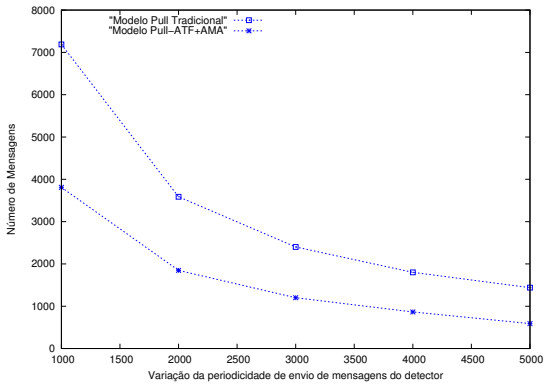
A figura 4(a) apresenta o experimento aplicando a estratégia AMA aos algoritmos *Pull* e *Push* comparados aos seus modelos tradicionais. Este experimento revela que o reaproveitamento da estratégia AMA está relacionada a proporção da periodicidade de envio da aplicação juntamente com a periodicidade de envio do algoritmo de detecção. Observe que para um Δ_i igual a 1 ms tem-se um ganho aproximado de 10% para ambos modelos. Conforme o Δ_i cresce a estratégia obtém resultados melhores, ou seja, para Δ_i igual a 5000 ms obtém-se um ganho de 46%. Em síntese, o melhor caso para a estratégia AMA é ter uma aplicação que envia mensagens em períodos menores que o Δ_i. Este caso pode ser visualizado na figura 4(b), onde fixou-se a periodicidade de envio do detector em 3000 ms e variou-se a periodicidade de envio da aplicação. Nos instantes em que a aplicação envia mensagens em períodos menores que o *FD* observa-se o melhor caso com um reaproveitamento de mensagens de 100%. Para instantes em que a aplicação envia mensagens em períodos superiores a periodicidade do *FD*, o experimento revela que o grau de reaproveitamento de mensagens se relaciona a quantidade de tempo em que se pode atrasar uma mensagem futuramente, ou seja, quanto mais próximo do instante em que o detector for gerar uma mensagem de controle ele reaproveitar uma mensagem melhor. Os experimentos com a estratégia ATF e ATF+AMA, aplicadas ao modelo *Pull* são apresentados na figura 5. Analisando a figura 5(a) pôde-se observar que em períodos com maior sobrecarga do detector, por exemplo,



(a) Ganho da estratégia ATF



(b) Ganho extra c/ a estratégia AMA



(c) Ganho da ATF+AMA

Figura 5. Número de mensagens enviadas variando o Δ_i, estratégia ATF e combinação ATF+AMA

Δ_i igual a 1000 ms, obteve-se os melhores resultados tendo um ganho aproximado de 45% no número de mensagens enviadas, ao passo que para um Δ_i igual a 5000 ms, tem-se um ganho de 38%. Este experimento indica que a estratégia ATF tem um melhor proveito em períodos de sobrecarga do canal.

Aplicando-se ambas estratégias (AMA e ATF) ao modelo *Pull*, observa-se (figura 5(b)) que a estratégia AMA consegue obter um ganho extra de 11,8% somado ao ganho da estratégia ATF. Combinando as duas estratégias obteve-se (figura 5(c)) os melhores resultados para a redução do número de mensagens, atingindo um ganho médio de 55%, se comparado ao modelo *Pull* tradicional.

Em síntese, os resultados mostram que as estratégias AMA e ATF reduzem o número de mensagens de controle se comparadas aos modelos tradicionais. Mas outro experimento fez-se necessário para avaliar o desempenho das estratégias propostas com o algoritmo Lazy o qual reaproveita mensagens das aplicações para suprir mensagens de controle. Assim, para avaliar o reaproveitamento de mensagens, foi utilizada uma aplicação sintética que troca mensagens no transcorrer de cada 10000 ms.

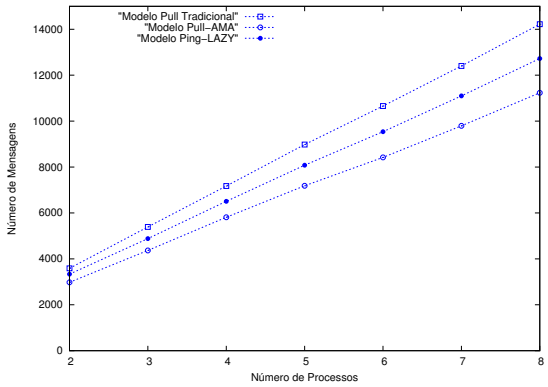
Analisando o gráfico 6(a) o qual apresenta uma comparação do modelo *Pull* tradicional, *Pull* AMA e o algoritmo Lazy, pode-se verificar que ambas variações do modelo *Pull* que reaproveitam mensagens das aplicações obtiveram êxitos se comparado ao modelo tradicional. Entretanto a estratégia AMA demonstrou-se mais eficiente garantindo um ganho aproximado de 11,4% de economia nas mensagens. Este ganho está agregado à ausência de confirmação para cada mensagem da aplicação enviada. Além disso, como a estratégia AMA contabiliza somente as mensagens emitidas pela aplicação no processo receptor, para estas mensagens o canal de comunicação não necessita ser confiável. O algoritmo Lazy, devido ao fato deste contabilizar as mensagens no emissor e por depender de uma confirmação, exige um canal confiável [6].

Outra diferença que vale ser salientada é que como apresentado no gráfico 6(b) a estratégia AMA pode ser extendida ao modelo *Push* inclusive, e para tal combinação foi obtido o melhor resultado entre todos os algoritmos experimentados. Entretanto o algoritmo Lazy somente pode ser aplicado a algoritmos que trabalham na forma de monitoramento bidirecional (*two-ways*).

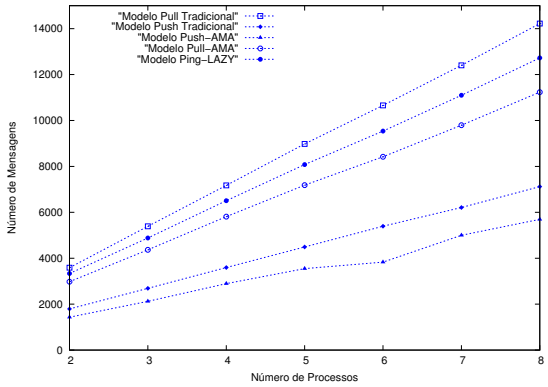
Em síntese, as estratégias ATF e AMA mostram-se eficientes para conter o problema da explosão de mensagens, sem exigir o uso de canal confiável.

4.2.2 Influência das Estratégias na QoS dos Detectores de Defeitos

Nesta seção são realizados experimentos utilizando as métricas propostas por Chen, Toueg e Aguilera [4]. Inicialmente calcula-se a métrica do T_D (tempo de detecção), para estes experimentos foram retiradas amostras referentes a ocorrência de suspeitas, dentro de períodos de 24 horas. Um *timeout* fixo com o valor de 4000 ms foi utilizado. A tabela 1 apresenta os tempos de detecção para execuções com os detectores *Pull* e *Push* nos estilos tradicional e uti-



(a) Lazy comparado ao modelo Pull



(b) Análise de todos os algoritmos

Figura 6. Comparação do número de mensagens de controle incluindo o algoritmo Lazy

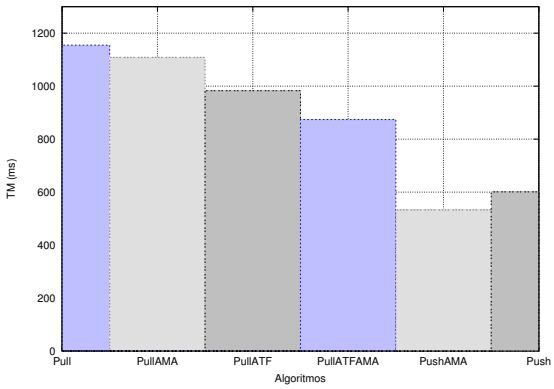
lizando as extensões com as estratégias AMA e ATF. Os resultados mostram que o tempo médio de detecção foi de $\overline{T_D} = 7305ms$ para o modelo *Pull* tradicional, ao passo que o pior resultado foi obtido com a estratégia AMA referente a $\overline{T_D} = 9903ms$. Com o modelo tradicional se obteve um tempo de detecção 26% menor se comparado a média entre o pior e melhor tempo obtido com as estratégias, isto implica num ganho médio de 2295 ms. O aumento no tempo de detecção do *Pull* pelo uso das estratégias já era esperado. Entretanto este não é o único fator que indica QoS.

Por outro lado, quando aplicada a estratégia AMA ao modelo *Push* e comparado ao seu respectivo modelo tradicional (tabela 1), observou-se que esta estratégia não piorou o tempo de detecção. A média obtida pelo modelo tradicional é de aproximadamente $\overline{T_D} = 5391ms$, o menor tempo se comparado com os experimentos utilizando o modelo *Pull*. Entretanto a média pertencente a estratégia AMA é de aproximadamente $\overline{T_D} = 5080ms$ o que corresponde a uma melhora de aproximadamente 5,8% se comparado ao seu correspondente na forma tradicional.

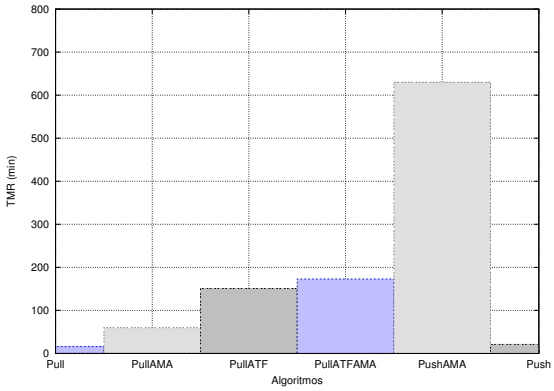
Tabela 1. Comparação do $\overline{T_D}$ para as extensões utilizando o modelo *Pull* e *Push*

	Algoritmos Utilizados					
Métrica	*Pull*	*PullAMA*	*PullATF*	*PullATFAMA*	*Push*	*PushAMA*
$\overline{T_D}$	7305	9903	9606	9201	5391	5080

Outros experimentos mostram que a precisão dos *FDs* utilizando a abordagem proposta podem ser melhoradas. Para tal foi calculado o T_M (tempo de duração do erro) e o T_{MR} (tempo para recorrência ao erro) de cada abordagem a uma periodicidade de envio de mensagens de Δ_i = 5000 ms. Nos experimentos ocuparam-se todos os processos distribuídos no Planet-Lab. Como resultado, o comportamento das estratégias propostas demonstraram resultados eficientes que podem ser visualizados na figura 7.



(a) Tempo do TM para todos os algoritmos



(b) Tempo do TMR para todos os algoritmos

Figura 7. Cálculo do T_M e do T_{MR}

Para o modelo *Pull* o menor $\overline{T_M}$ obtido foi com a combinação da estratégia ATF + AMA onde o valor médio observado é 24,3% menor do que o *Pull* tradicional. Já

para o modelo *Push* a estratégia AMA obteve um ganho relativo de 11% se comparada ao seu estilo tradicional. O bom desempenho das estratégias no que diz respeito ao T_M pode ser explicado pelos "tempos mínimos"obtidos. As estratégias AMA e ATF aplicadas ao modelo *Pull* conseguiram atingir valores mínimos de $T_M = 1ms$, enquanto que a estratégia AMA aplicada ao *Push* atingiu valores mínimos de $T_M = 3ms$.

Para os cômputos do $\overline{T_{MR}}$ o pior resultado obtido corresponde a 16,98 min atingido pelo modelo *Pull* tradicional, ao passo que a combinação de ATF + AMA obteve o melhor resultado para este mesmo modelo, chegando a média de 173,49 min o que equivale a um ganho aproximado de 921,73%. A estratégia AMA extendia ao modelo *Push* obteve o melhor resultado para todos os experimentos de aproximadamente $\overline{T_{MR}}$ = 630,65 min. Equivalente a um ganho aproximado de 2834,62% se comparado ao modelo *Push* tradicional. Em síntese pode-se dizer que a estratégia AMA aplicada ao modelo *Push* obteve o melhor resultado em termos de precisão, pois este apresenta o menor T_M e o maior T_{MR}. Logo este algoritmo obtém a maior rapidez para a recuperação de um erro e a menor taxa de recorrência ao mesmo.

5 Considerações Finais

Uma abordagem genérica para reduzir o número de mensagens de controle foi proposta neste trabalho. Os experimentos realizados mostraram a eficiência das estratégias em termos do número de mensagens de controle enviadas no canal de comunicação. Na busca pelos melhores resultados, a estratégia ATF destacou-se em períodos de maior sobrecarga no canal de comunicação. Já para as análises com a estratégia AMA observou-se que o ganho para reduzir o número de mensagens de controle está vinculado a periodicidade de envio do *FDs* em relação a periodicidade de envio da aplicação cliente. Em outras palavras, se a aplicação cliente enviar mensagens em períodos menores que o *FD*, tem-se 100% das mensagens de controle supridas (melhor caso). Em síntese, o grau de reaproveitamento de mensagens para ambas estratégias, diz respeito a quantidade de tempo em que se pode atrasar uma mensagem futuramente.

A estratégia AMA foi comparada ao algoritmo Lazy, em busca de avaliar o reaproveitamento de mensagens das aplicações pelos algoritmos de detecção. Para estes experimentos verificou-se que a estratégia AMA obteve êxito se comparada ao Lazy, seu principal competidos. Este êxito, basicamente está agregado à ausência de confirmação para cada mensagem da aplicação enviada. Nas análises da influência das estratégias propostas na QoS dos detectores, embora tenha verificado-se um aumento para o T_D em relação a estratégia ATF, de modo geral, observou-se que as estratégias contribuem para o aumento da precisão dos detectores, obtendo significativas melhoras para os cálculos do T_M e T_{MR}, o que resulta numa melhor relação custo/benefício.

Referências

[1] M. Bertier, O. Marin, and P. Sens. Performance analysis of a hierarchical failure detector. In *DSN*, pages 635–644, 2003.

[2] M. W. Burns, A. D. George, and B. A. Wallace. Simulative performance analysis of gossip failure detection for scalable distributed systems. *Cluster Computing*, 2(3):207–217, 1999.

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

[4] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Trans. Comput.*, 51(1):13–32, jan 2002.

[5] P. Felber, X. Défago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'99)*, pages 132–141, Washington, USA, Sept. 1999. IEEE Computer Society.

[6] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *PRDC '01: Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing*, pages 146–153, Washington, DC, USA, dec 2001. IEEE Computer Society.

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

[8] F. C. Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26, 1999.

[9] N. Hayashibara, A. Cherif, and T. Katayama. Failure detectors for large-scale distributed systems. In *SRDS '02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, page 404, Washington, USA, 2002. IEEE Computer Society.

[10] M. Larrea, S. Arévalo, and A. Fernández. Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 34–48, London, UK, 1999.

[11] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus (brief announcement). In *SRDS*, pages 52–59, New York, NY, USA, 2000. ACM Press.

[12] L. Peterson, A. Bavier, M. Fiuczynski, S. Muir, and T. Roscoe. Towards a Comprehensive PlanetLab Architecture. Technical report, PlanetLab Consortium, June 2005.

[13] N. Sergent, X. Défago, and A. Schiper. Impact of a failure detection mechanism on the performance of consensus. In *Proc. IEEE Pacific Rim Symp. on Dependable Computing (PRDC)*, Seoul, Korea, 2001.