Projeto e Análise de Desempenho de um Protocolo de Difusão Atômica Personalizado*

Fabíola Gonçalves Pereira Greve[1] **, Jean-Pierre Le Narzul , Xiaojun Ma**[2]

[1]Departamento de Ciência da Computação
Universidade Federal da Bahia
Campus de Ondina, 40170-110 Bahia, Brasil

[2]ENST Bretagne and IRISA
Campus de Beaulieu, 35042 Rennes Cedex, France

fabiola@ufba.br, {lenarzu,mxiaojun}@irisa.fr

Resumo. *A comunicação entre as réplicas de um serviço replicado precisa ser orquestrada por uma primitiva de* difusão atômica *a fim de garantir o estado consistente das mesmas. A implementação de uma tal primitiva é desta maneira um fator crítico para o bom desempenho desse tipo de serviço. Com o intuito de conceber uma solução tão eficiente quanto modular, propomos neste artigo um protocolo de difusão atômica, obtido a partir de uma especialização de um serviço genérico de acordo, que apresenta as seguintes características inovadoras: ele possui mecanismos para lidar diretamente com a perda de mensagens da aplicação e implementa a entrega atômica das mesmas sem recorrer ao uso de uma primitiva de* difusão confiável*. Uma prova da correção do protocolo é fornecida e alguns detalhes relativos à sua implementação são apresentados. O seu desempenho foi ainda avaliado a partir da realização de testes experimentais, segundo a variação de alguns parâmetros significativos.*
Palavras-Chave: Sistemas Distribuídos, Replicação Ativa, Difusão Atômica, Consenso

Abstract. *This work provides an efficient and realistic atomic broadcast protocol which supports the loss of network messages sent by clients. It is designed as a specialization of a general agreement framework. As far as we know, there is no other consensus-based protocol proposed in the litterature that deals directly with losses without using the reliable broadcast primitive as a resource to deliver messages in a reliable manner. We give details regarding its implementation and correctness proof. Moreover, we analyze its performance through the run of a number of experimental tests.*
Keywords: Distributed Systems, Active Replication, Atomic Broadcast, Consensus

1. Introdução

Uma maneira clássica de tornar um servidor confiável consiste em replicá-lo em diferentes máquinas de um sistema distribuído. O estado do servidor é compartilhado entre as réplicas que executam ações coordenadas a fim de implementar o serviço requerido. Se as máquinas falham de maneira independente, a invocação do serviço pelo cliente será bem sucedida mesmo se algumas das réplicas falham antes de terminar as ações requeridas. Na técnica da *replicação ativa* [19] todas as réplicas têm o mesmo papel. Para preservar o estado coerente do servidor, uma primitiva de *difusão atômica* [12] deve ser usada a fim de garantir que as mensagens provenientes dos clientes sejam entregues numa mesma ordem total ao grupo de servidores.

Neste artigo, apresentamos um protocolo de difusão atômica obtido a partir de uma redução a um serviço genérico de consenso, de nome GAF (*General Agreement Framework*) [13]. O problema do consenso é um denominador comum entre diversos problemas práticos presentes na concepção de sistemas tolerantes a faltas. As soluções baseadas no consenso são atrativas, tanto do ponto de vista prático quanto teórico, pois além do caráter modular e elegante, exibem uma caracterizarão precisa das propriedades de *liveness* (vivacidade) e *safety* (precisão) ligadas aos problemas.

O protocolo de difusão atômica proposto admite perda de mensagens da aplicação e implementa diretamente os mecanismos necessários para a sua recuperação. Ao nosso conhecimento, nenhum outro protocolo baseado no consenso, desenvolvido até então [3, 1, 16, 15, 10, 11, 17], fornece mecanismos

*Este trabalho foi realizado com financiamento do CNPQ/Brasil e projeto de cooperação CAPES-COFECUB (497/05).

para lidar diretamente com tais perdas. Todos eles se fundamentam na existência de canais de comunicação confiáveis ou se baseiam no uso de uma primitiva de *difusão confiável* [12] para assegurar a entrega atômica das mensagens. A implementação de tal primitiva tem um alto custo: para cada mensagem proveniente do cliente e recebida no grupo, $O(n^2)$ mensagens são retransmitidas às n réplicas do servidor. Trabalhos foram propostos com o intuito de diminuir o custo de implementação da difusão confiável [18]. Nosso interesse, entretanto, é o de eliminar o seu uso na implementação da difusão atômica propriamente dita.

O protocolo proposto foi implementado e testes experimentais foram realizados com o intuito de medir o tempo médio de entrega de uma requisição emitida por um cliente à camada de aplicação. Foram considerados experimentos que verificaram o comportamento do protocolo segundo a sua escalabilidade (variação da quantidade de processos no grupo), a duração de cada rodada de consenso e a existência de falhas. As características do protocolo desenvolvido, aliadas às facilidades do framework GAF (onde, por exemplo, diversas mensagens podem ser ordenadas ao mesmo tempo), proporcionam um menor impacto na latência de entrega das mensagens à medida que o tamanho do grupo aumenta.

Nas próximas seções, descrevemos inicialmente o modelo de replicação considerado (seção 2), em seguida fornecemos uma descrição sucinta dos parâmetros necessários à utilização do framework GAF (seção 3). Posteriormente, na seção 4, descrevemos o protocolo de difusão atômica proposto e apresentamos detalhes relativos à sua implementação. Na seção 5, mostramos resultados obtidos através de testes experimentais. Finalmente, na seção 6, concluimos o trabalho e apresentamos no anexo a prova de correção do protocolo.

2. Modelo de Sistemas

Designamos por S um servidor único particular. Para tolerar falhas por parada (do inglês, *crash*) [3, 6], o servidor é replicado em n processos: cada processo p_i executa uma réplica de S. $\Pi = \{p_1, ..., p_n\}$ é então visto como o grupo de processos associados ao servidor S. Nesse modelo de falhas, um processo está ativo ou inativo. Inicialmente, todos os processos estão ativos e se comportam de acordo com a sua especificação. Quando um processo falha, torna-se inativo e mantém-se nesse status durante todo o resto da execução do sistema. Um processo que nunca está inativo é considerado *correto*; de outra maneira, ele é considerado um *processo faltoso*. Consideramos que existe uma maioria de processos corretos no sistema. Seja f o número de processos faltosos no sistema, então a relação $f < n/2$ prevalece.

Os processos se comunicam e cooperam pela emissão de mensagens através de canais de comunicação segundo um modelo de comunicação assíncrono. Nenhuma hipótese temporal existe no que diz respeito à realização das ações efetuadas pelos processos ou pelos canais. As entidades do modelo evoluem segundo a sua própria velocidade de execução. Os canais não criam, não alteram, nem duplicam as mensagens que ali trafegam. Entretanto, eles admitem perdas de mensagens de forma eqüitativa. Ou seja, se um processo p_i envia a um processo correto p_j uma mensagem m uma infinidade de vezes, então p_j recebe m uma infinidade de vezes. Tal modelo de falhas é conhecido em inglês como *fair-lossy* [2].

2.1. O Problema do Consenso

Informalmente o problema do consenso [6, 3] é definido da seguinte maneira: cada processo correto p_i propõe um valor v_i e todos os processos corretos devem "decidir"por um único valor v, escolhido dentre aqueles que foram propostos. Formalmente, o consenso é definido pelo seguinte conjunto de propriedades [6, 3].

- C_Terminação: todo processo correto decide de maneira definitiva;
- C_Integridade: um processo decide no máximo uma vez;
- C_Validade: se um processo decide v, então v foi proposto por algum processo;
- C_Acordo_Uniforme: dois processos (corretos ou não) não decidem diferentemente.

Esse problema fundamental não tem solução determinista num sistema assíncrono, mesmo em presença de uma única falha [6]. Uma das estratégias adotadas para contornar tal resultado de impossibilidade consiste em estender o modelo assíncrono com algum grau de "sincronismo". Com este intuito, um dos avanços mais significativos é a proposta de uso dos *detectores de falhas não confiáveis*. Informalmente, um detector de falhas é um conjunto de "oráculos"distribuídos que fornece dicas aos processos sobre quais deles estão falhos. Chandra e Toueg [3] propuseram um algoritmo que resolve o problema do consenso baseado num detector de falhas do tipo $\diamondsuit\mathcal{S}$ e à condição que uma maioria de processos seja correta. A classe de detectores $\diamondsuit\mathcal{S}$ (do inglês, *eventually strong*) garante que todo processo falho será finalmente suspeito por todos os processos corretos; além disso, existirá um instante a partir do qual algum processo correto não será considerado suspeito por nenhum outro processo correto.

2.2. Difusão Atômica

Informalmente, um serviço de difusão atômica (*atomic broadcast*) assegura que os processos de um grupo entregarão um mesmo conjunto de mensagens e na mesma ordem total. Formalmente, esse serviço é definido da seguinte maneira [3]:

- AB_Terminação: se um processo correto *envia* uma mensagem m então todos os demais processos corretos *entregam* m;
- AB_Integridade: um processo só *entrega* uma mensagem no máximo uma vez;
- AB_Acordo_Uniforme: se um processo *entrega* uma mensagem m então todos os demais processos corretos também entregam m;
- AB_Validade: se um processo *entrega* uma mensagem m então algum processo *enviou* m;
- AB_Ordem_Total: se dois processos p_i e p_j *entregam* as mensagens m e m', então p_i entrega m antes de m', se e somente se, p_j entrega m antes de m';

As quatro primeiras propriedades acima caracterizam a primitiva de difusão confiável (*reliable broadcast*) [12]. Ela garante que uma mensagem enviada será entregue por todos os processos corretos ou por nenhum deles.

3. GAF: um Framework para Derivação de Protocolos de Acordo

GAF (do inglês, G*eneral* A*greement* F*ramework*) [13] é um framework genérico para a realização de um acordo a partir do protocolo clássico de Chandra e Toueg [3]. Ele dispõe de alguns parâmetros genéricos que devem ser instanciados de maneira estática (em *tempo de compilação*) para a geração automática dos protocolos. Os processos que desejam construir uma lógica de acordo particular devem instanciar estes parâmetros com valores adaptados à semântica do problema. A seguir, descrevemos sucintamente os principais parâmetros de GAF necessários à instanciação da difusão atômica. Maiores detalhes sobre o uso dos parâmetros e funcionamento do framework podem ser obtidos em [13].

- GET: graças a esta função a camada da aplicação irá transmitir ao framework as proposições para o acordo. Durante a execução do protocolo de acordo, esta função poderá ser chamada diversas vezes. Assim, um processo pode mudar seu valor de proposição sempre que ele desejar. Isto permite, dentre outros, que valores cada vez mais significativos sejam propostos sem necessidade de esperar pelo fim de um consenso.
- $\prec$: relação que estabelece se a proposição de um processo contribui com mais informações; em outras palavras, se ela é mais significativa que uma proposição anterior.
- $\mathcal{F}$: função aplicada sobre o conjunto de valores propostos (de entrada) e cujo objetivo é o cálculo do valor de decisão (de saída). Ela é aplicada quando uma quantidade suficiente de proposições foram recolhidas ao longo do consenso.
- ACCEPTABLE: predicado cujo objetivo é a verificação da aceitação do valor escolhido (decisão efetuada). Graças a ele, um processo pode participar a um consenso sem que ele possua um valor de proposição significativo.

4. Protocolo de Difusão Atômica

Grande parte dos protocolos de difusão atômica propostos a partir de uma redução a um serviço de consenso [3, 1, 16, 15] utilizam canais confiáveis. Na prática, a abstração de canais confiáveis exige meios de comunicação seguros nos quais a mínima perda de mensagens da aplicação torna-se inaceitável. Protocolos propostos mais recentemente admitem a perda de mensagens [10, 11, 17]. Entretanto, todos eles se apoiam no uso de uma primitiva de difusão confiável [12] para propagar internamente ao grupo de processos as requisições provenientes dos clientes. Quando um processo recebe uma mensagem m pela primeira vez, ele difunde m aos demais membros do grupo e somente depois é que ele entrega m localmente. Tal primitiva garante que uma mensagem enviada ao grupo será entregue por todos os processos corretos ou por nenhum deles (atomicidade na entrega). Sua implementação é entretanto custosa: para cada mensagem enviada pelo cliente, $O(n^2)$ mensagens serão transmitidas ao grupo. Como na prática, a maioria das mensagens não se perdem, o uso sistemático de tal primitiva custosa deveria e pode ser evitado.

Com o objetivo de conceber uma solução tão realista quanto eficaz, apresentamos um protocolo de difusão atômica que admite a perda de mensagens de maneira eqüitativa, ou seja, estaremos considerando canais do tipo *fair-lossy*. Além disso, diferentemente de todos os demais protocolos, não faremos uso da primitiva de difusão confiável para difundir as mensagens recebidas no grupo. Um mecanismo de retransmissão de mensagens específico, que utiliza as facilidades do próprio serviço de consenso, se encarregará de garantir por um lado, a atomicidade da entrega das mensagens e, por outro lado, o re-envio das mensagens somente aos processos que não as possuem.

4.1. Descrição do Protocolo

Interface com a aplicação. Supomos que os clientes geram um fluxo contínuo de requisições através de alguma primitiva de difusão ao grupo de réplicas. O controle do recebimento de tais requisições deve ser feito por um módulo específico do lado do cliente. Nesse caso, semânticas do tipo *no mínimo uma vez*, *no máximo uma vez*, etc. deverão ser adotadas. A implementação de um tal controle está fora de escopo desse trabalho.

O processo servidor que recebe uma mensagem m do cliente, invoca internamente a função A_BROADCAST(m) para dar início ao protocolo que difunde m atomicamente no grupo. Os processos entregam as mensagens localmente por intermédio da função A_DELIVER(), chamada pela camada da aplicação responsável pela execução das requisições.

Princípio. A ordenação das mensagens da aplicação provenientes dos clientes, é realizada passo a passo pelo protocolo. A cada passo, várias novas mensagens são ordenadas graças ao consenso. Por motivos de eficiência, somente as identidades das mensagens são passadas para o acordo. Para dar início a um novo consenso, espera-se o fim do anterior. Cada nova seqüência de mensagens ordenadas estende a seqüência global das mensagens anteriormente observadas. A fim de garantir a entrega atômica das mensagens emitidas pelos clientes (propriedade AB_Acordo_Uniforme) somente mensagens propostas por uma "maioria"de processos estarão sendo ordenadas. Isto porque, em caso de perda, garante-se que existirá ao menos um processo correto que poderá retransmitir a mensagem. À demanda da aplicação, através de A_DELIVER(), o protocolo entrega a primeira mensagem ordenada que esteja disponível. Os processos que ainda não receberam da rede as mensagens já ordenadas esperam que elas sejam recebidas antes de entregá-las à aplicação. Um mecanismo de retransmissão de mensagens foi incorporado para evitar que os processos fiquem bloqueados à espera de mensagens perdidas.

Funcionamento. O protocolo está ilustrado pela figura 1. Cada processo p_i controla localmente as seguintes variáveis:

- k: o número do consenso atual;
- $Received_i$: conjunto de mensagens recebidas dos clientes;
- $A\_Delivered_i$: conjunto de identidades de todas as mensagens ordenadas dentro do grupo;
- $Undelivered_i$: conjunto de identidades das mensagens já ordenadas mas que ainda não foram entregues à aplicação. Neste conjunto, as mensagens são classificadas primeiramente em função do número do consenso onde elas foram decididas e em seguida pela aplicação de uma função determinista, conhecida a priori por todos os participantes (e.g. a identidade das mensagens).

O protocolo é composto de dois *threads* concorrentes: AB1 et AB2. O *thread* AB1 é responsável pela interface com a aplicação e com a camada de comunicação. O *thread* AB2 efetua a ordenação das mensagens com ajuda do protocolo GAF e atualiza o estado local do processo a partir das decisões tomadas.

Thread AB1: interface com a aplicação e com a camada de comunicação. Toda mensagem recebida (pela invocação da operação A_BROADCAST(m)) será diretamente armazenada no conjunto $Received_i$ (linhas 2-3). Diferentemente dos protocolos clássicos de difusão atômica, as mensagens recebidas dos clientes não são difundidas no início do protocolo por uma primitiva de difusão confiável.

Quando a operação A_DELIVER() é solicitada (linha 4), a primeira mensagem no topo da fila $Undelivered_i$ é entregue para a aplicação; esta entrega só é efetuada após a efetiva recepção da mensagem por parte do processo (linhas 5-6).

O processo que tenha recebido REQUEST_MSG(p_j, m) (linha 7) retransmite a mensagem m ao processo p_j que a solicitou através do envio de A_MSG(m) (linha 8). O processo que recebe esta mensagem A_MSG(m) (linha 9) armazena diretamente m em $Received_i$ (linha 10).

Thread AB2: ordenação das mensagens. Cada processo p_i lança, em *background*, o serviço de acordo GAF (linha 13) para o cálculo do novo conjunto de mensagens. Identificamos pela variável k cada consenso efetuado, sabendo-se que um consenso de número k não começa antes que o consenso anterior (de número $k-1$) tenha terminado. O processo inicia ou participa do acordo k somente quando ele possui um valor significativo a propor. Cada valor representa as mensagens recebidas localmente, mas ainda não ordenadas: a diferença entre os conjuntos $Received_i$ e $A\_Delivered_i$. Este valor é o resultado retornado pela função GET() (linhas 16-17) do protocolo GAF. Quando o acordo k termina, o processo atualiza os subconjuntos locais a partir do novo conjunto de mensagens decidido ($Decided^k$) pela chamada ao procedimento Entrega-Confiável (linha 14).

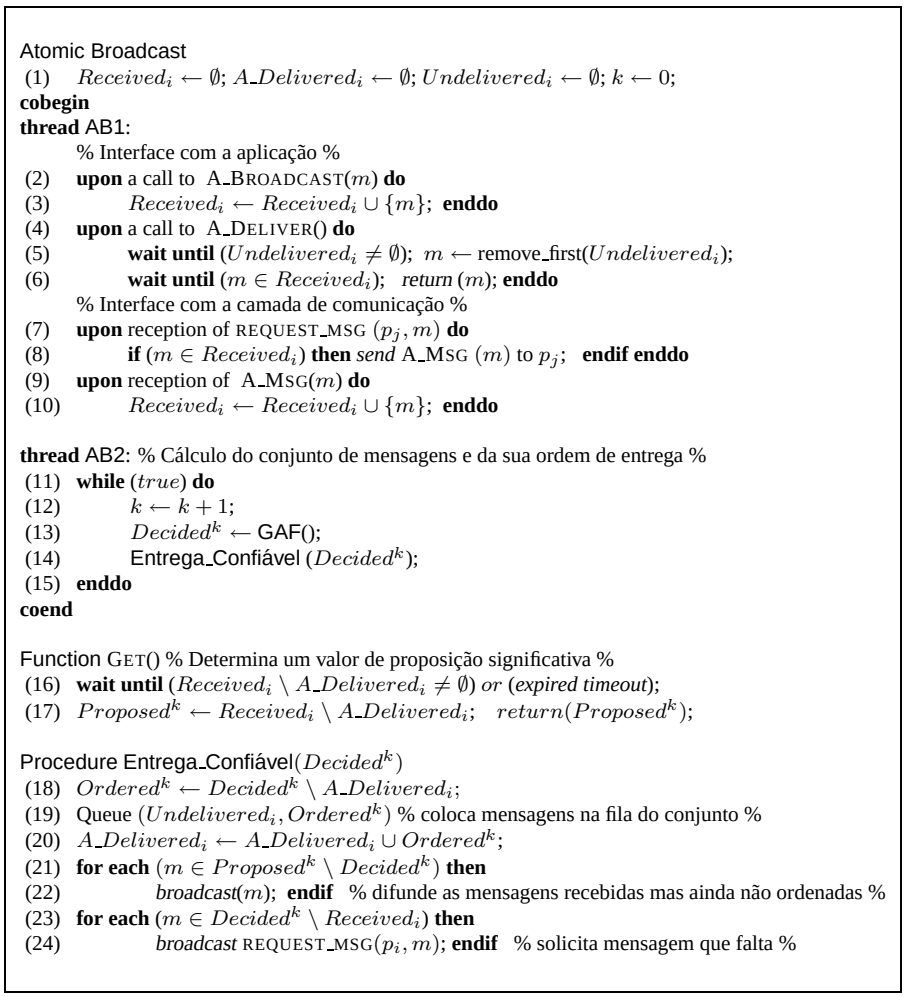

Figura 1: Protocolo de Difusão Atômica

O procedimento Entrega-Confiável As mensagens de $Decided^k$ que ainda não foram entregues (linha 18) são adicionadas à fila $Undelivered_i$ (linha 19) e em seguida ao conjunto $A\_Delivered_i$ (linha 20). Devido à possibilidade de perda de mensagens, algumas das mensagens podem ser recebidas somente por alguns processos. Dois casos devem ser então considerados:

• Caso a [um processo recebeu do cliente uma mensagem que os outros ainda não receberam] – a cada vez que uma decisão é tomada, o processo p_i verifica localmente se a decisão levou em conta todo o subconjunto de mensagens que ele havia proposto ao consenso. Caso esta condição não se verifique ($m \in Proposed^k$ e $m \notin Decided^k$), p_i difunde a mensagem do cliente ao grupo por intermédio da primitiva não confiável *broadcast*(m).

• Caso b [um processo não recebeu do cliente a mensagem que outros receberam] – quando um processo p_i verifica que uma mensagem foi ordenada sem que ele a tenha recebido ($m \in Decided^k$ e $m \notin Received_i$), ele solicita ao grupo, pela emissão de REQUEST_MSG(p_i, m), a mensagem que lhe falta. Na prática, esta solicitação ao grupo pode ser substituída por uma protocolo de requisição ponto-a-ponto até que a mensagem m seja obtida por p_i. Como supõe-se uma maioria de processos corretos, p_i terminará por receber m.

4.2. Estabilização das Mensagens no Grupo

O processo de estabilização de mensagens atende a dois objetivos: i) retransmissão e recuperação de mensagens perdidas e ii) eliminação local de *mensagens estáveis*, i.e., mensagens que tenham sido recebidas da rede por todos os processos do grupo. Nesse último caso, evita-se o crescimento infinito dos conjuntos de mensagens utilizados pelo protocolo; particularmente, do conjunto $A\_Delivered_i$.

Através do procedimento Entrega_Confiável, a rotina de estabilização utiliza as informações provenientes do próprio acordo para evitar a retransmissão inútil de mensagens. Inicialmente, um processo que recebe uma mensagem do cliente espera o fim do próximo acordo para então retransmiti-la aos outros membros (no caso da mensagem não ter sido ordenada). Para as mensagens que já foram ordenadas, elas só serão re-enviadas, sob demanda, aos processos que não as receberam da rede.

As mensagens ordenadas são entregues à aplicação sem necessidade de esperar que elas sejam estáveis no grupo. Sabe-se, entretanto, que ao menos uma maioria de processos as possuem, pois o protocolo adia as ordenações até que uma maioria de processos as tenham recebido da rede. Como supõe-se uma maioria de corretos, mensagens não estáveis poderão sempre ser recuperadas. Para coletar informações relativas à estabilização das mensagens, os processos difundem e atualizam periodicamente um *vetor de relógios* [14] com n posições. Cada posição i do vetor contem o identificador em ordem linear crescente da mensagem mid_i mais recentemente entregue à aplicação pelo processo p_i. O valor de mid_i é atualizado cada vez que a linha 6 do protocolo é executada. Seja mid_k o menor valor de identificador presente no vetor. A cada instante, mensagens com identificadores menores ou iguais a mid_k são consideradas estáveis no grupo; consequentemente, elas poderão ser descartadas localmente.

4.3. Definição dos Parâmetros para o Framework GAF

As principais funções de GAF para resolver a difusão atômica são apresentadas na tabela 1 e são descritas a seguir.

• A função $\mathcal{F}$: a escolha desta função é crucial para a satisfação das propriedades definidas para o problema: AB_Acordo_Uniforme e AB_Terminação. Para garantir o acordo é importante considerar no cálculo da decisão as mensagens propostas por ao menos uma "maioria"de processos. Dado que supostamente uma maioria é correta (condição imposta para permitir a resolução do consenso) então ao menos um processo, dentre aqueles que possuem a mensagem, será correto e poderá retransmiti-la futuramente aos outros processos que ainda não a receberam. Assim, a função $\mathcal{F}$ é definida como sendo a *intersecção* das proposições coletadas.

• A função GET() – Retorna o conjunto de mensagens recebidas localmente pelo processo, mas que ainda não foram ordenadas: $Received_i \setminus A\_Delivered_i$ (linhas 16-17). É importante observar que, durante a execução do consenso GAF, enquanto uma decisão ainda não tiver sido tomada, essa função poderá ser chamada diversas vezes. Assim, num mesmo consenso, o processo poderá incrementar o conjunto de mensagens proposto para ordenação.

Devido à possibilidade de perda de mensagens, alguns processos podem ter recebido mensagens provenientes do cliente, enquanto outros não as receberam. Assim, alguns processos darão início ao consenso enquanto outros restarão bloqueados à espera de um valor de entrada significativo ($\neq \emptyset$). Para evitar tal bloqueio do protocolo GAF, autorizamos valores de proposição não significativos ($= \emptyset$). Isto é necessário, pois a chamada ao procedimento de retransmissão de mensagens perdidas (linhas 21-22), que poderia eventualmente desbloquear GAF, se faz somente quando uma decisão é tomada. O framework GAF autoriza assim um processo a propor inicialmente um conjunto vazio ($v = \emptyset$) e posteriormente a completar este valor inicial à medida que novas mensagens chegam ao processo durante a execução do protocolo. Vale ressaltar que se proposições de conjuntos vazios são freqüentemente emitidas, podemos comprometer a qualidade do acordo realizado; isto é, valores não significativos serão decididos. Uma só proposição vazia irá tornar inútil o cálculo da decisão efetuada pela aplicação da função $\mathcal{F}$. Para evitar tal situação, antes de propor um valor não significativo, os processos deverão esperar pela expiração de um valor de *timeout*. Este valor poe der definido em função do tempo estimado de transmissão de uma mensagem na rede; como ele é utilizado localmente, os processos poderão modificá-lo em função da percepção que possuam do comportamente da rede de comunicação.

• A função ACCEPTABLE(v) – Ela retorna $verdadeiro$ quando o valor v não é o conjunto vazio $\emptyset$ ou após a expiração de um timeout. Sua aplicação é importante, pois a função $\mathcal{F}$, definida anteriormente, pode gerar um valor $\emptyset$. Se isso acontece, este valor será rejeitado pelos processos e o protocolo continuará a ser executado até que uma decisão válida ($\neq \emptyset$) possa ser tomada. Caso uma decisão válida não tenha sido tomada até a expiração do timeout, a função retorna $verdadeiro$, mesmo quando o valor $v = \emptyset$. Esta medida foi tomada porque, devido à possibilidade de perda de mensagens, mais do que uma maioria de processos podem não ter recebido mensagens provenientes do cliente. Se isso ocorre, pela aplicação de $\mathcal{F}$, o valor de decisão será sempre $\emptyset$. Nesse caso, com base no procedimento de recuperação de mensagens adotado, é preciso que o consenso termine para que os processos possam recuperar as mensagens perdidas. Assim, após a expiração do timeout, o consenso está autorizado a tomar decisões não-significativas.

Parâmetro	Descrição
GET	Retorna ($Received_i \setminus A\_Delivered_i$) ou (*expiração de timeout*)
$\prec$	$\perp = \emptyset$; $\quad \forall v', \forall v, v' \prec v \iff v' \subset v$
$\mathcal{F}$	Retorna intersecção de todos os conjuntos de entrada: $\bigcap Proposed_i$
ACCEPTABLE(v)	Retorna (V se $v \neq \emptyset$) ou (*expiração de timeout*

Tabela 1: Parâmetros GAF para a Difusão Atômica com Suporte à Perda de Mensagens

4.4. Protocolo de Consenso de GAF

O protocolo utilizado na implementação de GAF foi o algoritmo sugerido por Chandra e Toueg (CT) em [3] com algumas modificações para que o seu funcionamento siga o modelo probabilístico proposto por [7] a fim de obter um consenso com garantias de qualidade de serviço. A seguir, o modelo considerado é apresentado e uma breve descrição do princípio do protocolo é fornecida. O leitor interessado em saber detalhes sobre o funcionamento desse protocolo, deve se reportar a [7].

Modelo Probabilístico. O modelo assíncrono é estendido com o seguinte grau de sincronismo. No lugar dos detectores de falhas não confiáveis do tipo $\diamondsuit\mathcal{S}$, supomos que cada processo p_i tem acesso a um relógio local cujos desvios são negligenciáveis. Os relógios são sincronizados através de um algoritmo de sincronização clássico [4]. Os tempos de transmissão das mensagens no sistema obedecem a uma função de repartição $\mathcal{F}$ cuja probabilidade de perdas é bem determinada. A função de repartição $\mathcal{F}$ mapeia $\Re^+$ em $[0, 1]$. Isso significa que quando p_i envia uma mensagem m para p_j, então m alcança p_j após x unidades de tempo, com uma probabilidade igual a $\mathcal{F}(x)$.

Princípio. Tal como no clássico CT, o protocolo implementado baseia-se no paradigma do "coordenador rotativo" e prossegue o cálculo em rodadas. A decisão se faz por bloqueio do valor decidido a um determinado instante. Cada rodada é guiada por um coordenador predeterminado escolhido segundo uma função determinística a partir da identidade dos processos participantes ao consenso. Um cálculo probabilista permite determinar a duração de cada rodada antes do início do protocolo. Consequentemente, durante a execução de uma rodada, o coordenador tem um tempo pré-definido para efetuar o seu cálculo, coletar as mensagens e decidir. As rodadas iniciam-se e terminam sempre num tempo pré-determinado. Caso uma decisão não seja efetivada, os processos esperam pelo início da nova rodada para dar continuidade ao cálculo. Se uma decisão ocorre antes do término de uma rodada, os processos ressincronizam o relógio e passam para a execução de um novo consenso. Nesse protocolo, a todo momento, as suspeitas de processos são substituídas pela expiração de um valor de *timeout*, que corresponde ao valor de duração de cada rodada.

5. Análise de Desempenho

O protocolo descrito anteriormente foi implementado como uma coleção de classes em linguagem JAVA 1.4. Testes foram realizados com o intuito de avaliar a *latência* necessária para a entrega atômica das requisições feitas pelo cliente. Esse valor corresponde à diferença entre o instante de tempo em que a requisição é emitida pelo cliente e o instante de tempo em que ela é entregue (pela réplica) à aplicação a fim de ser executada. O que se vê nos gráficos é o tempo de latência médio, que corresponde à média dos tempos obtidos a partir dos sucessivos testes, obtidos de todas as réplicas do grupo.

O desempenho do protocolo de difusão atômica depende fortemente do protocolo de consenso subjacente. Como visto anteriormente, o protocolo de consenso utilizado funciona por intermédio de rodadas com duração pré-determinada. Esse valor é definido pelo parâmetro TC (timeout de uma rodado do consenso). Em função de TC, outros timeouts do protocolo foram escolhidos. Os timeouts para a função GET() e para o predicado ACCEPTABLE() assumiram o mesmo valor que TC. Testes foram realizados com o intuito de escolher os melhores valores de TC para demandas de requisição a intervalos controlados. Além disso, mediu-se a variação do desempenho do protocolo de acordo com o crescimento do tamanho do grupo e com a ocorrência de falhas. Esses diversos cenários são detalhados em seguida.

Configuração do Sistema. Consideramos a seguinte configuração. Um cliente, hospedado numa máquina distinta, envia requisições a intervalos regulares, para um grupo de réplicas distribuídas. Foi usado um parque de máquinas Sun Ultra-5 (512 MB RAM) numa rede local Ethernet a 10 Mbit/s, sob condições normais de carga (XWindows, browsers, editores, etc.). As requisições foram feitas pelo cliente através do protocolo *IP-Multicast*. Por sua vez, as requisições dos clientes foram difundidas internamente pelos membros do grupo através dos protocolos *IP-Multicast* e *UDP-IP*.

Cenário 1: Timeout de Consenso X Tamanho do Grupo. Nesse cenário, analisamos o comportamento do protocolo de difusão segundo dois parâmetros: i) variações no número de processos no grupo (NP) e ii) variações nos valores de timeout do consenso (TC). Foram realizados cerca de mil testes; sendo que em cada um deles, o cliente emitiu 200 requisições, a intervalos regulares de $400ms$ entre cada uma.

A figura 2 mostra os resultados obtidos. O gráfico apresenta o tempo médio de entrega das requisições. Como esperado, a latência de entrega cresce com o aumento da quantidade de membros no

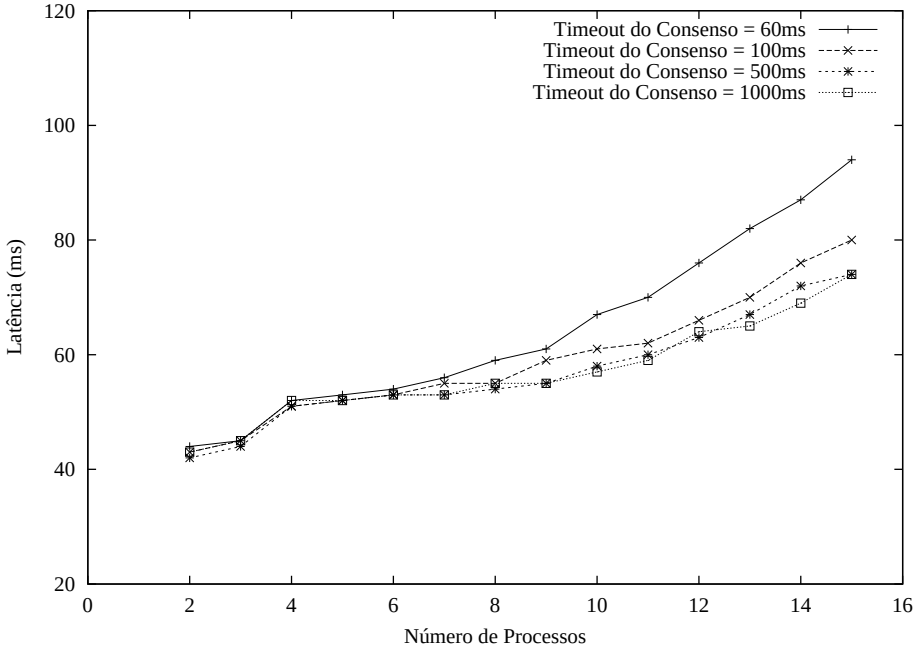

Figura 2: Tempo de entrega com variação no tamanho do grupo

grupo (NP). Além disso, todas as curvas apresentam a mesma evolução, idependente da variação dos timeouts de consenso. Com valores de TC muito pequenos, aumenta a latência de entrega das mensagens. Isso mostra que valores de TC pequenos ($< 100ms$) não são suficientes para decidir numa mesma rodada; sendo necessárias a realização de mais rodadas. Aparentemente, valores maiores de TC são suficientes para decidir na mesma rodada. Isso não significa que quanto maior o TC melhor é a latência. Na realidade, para um determinado valor de NP, o tempo médio de entrega mantém-se estável com o aumento do TC. Esse resultado pode ser auferido da figura 3. Quando o NP está entre 2 e 14, para valores de TC maiores que $300ms$, a latência é aproximadamente a mesma. É interessante observar que o protocolo GAF, através dos seus parâmetros, permite que diversas mensagens sejam decididas ao mesmo tempo; nesse caso, valores maiores de TC podem favorecer tais decisões conjuntas o que pode estar contribuindo para o melhor desempenho do protocolo.

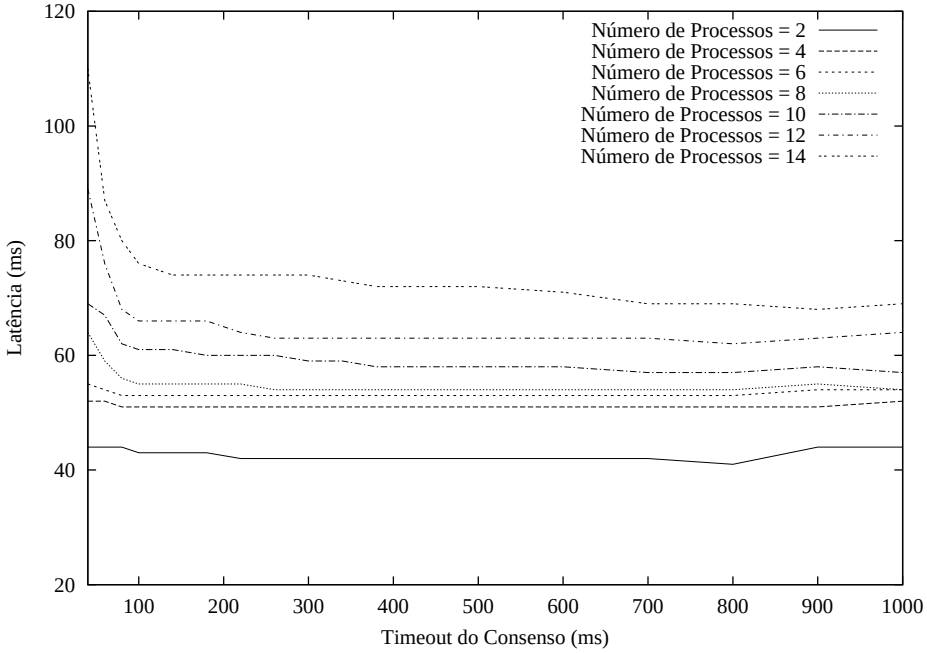

Figura 3: Tempo de entrega com variação do timeout do consenso

Cenário 2: Intervalo entre as Requisições X Tamanho do Grupo. Verificamos o comportamento do protocolo com a variação do tempo de emissão das requisições. Foram realizados cerca de mil testes; sendo que em cada um deles, o cliente emitiu 200 requisições. Adotou-se um TC de $1000ms$. Como mostra a figura 4, à medida em que o intervalo entre as requisições aumenta, a latência melhora gradualmente. Em compensação, quando o intervalo de emissão é muito pequeno, a latência média é muito superior e cresce rapidamente. Isso pode estar acontecendo porque uma quantidade maior de requisições num intervalo de tempo pequeno está exigindo uma maior quantidade de recursos, o que compromete o desempenho do protocolo como um todo. Inclusive, a sincronização exigida pelo consenso subjacente pode estar sendo afetada. Há também a possibilidade de estar havendo perda de mensagens e, nesse caso, o custo que o protocolo demanda para suportar tais perdas está sendo alto.

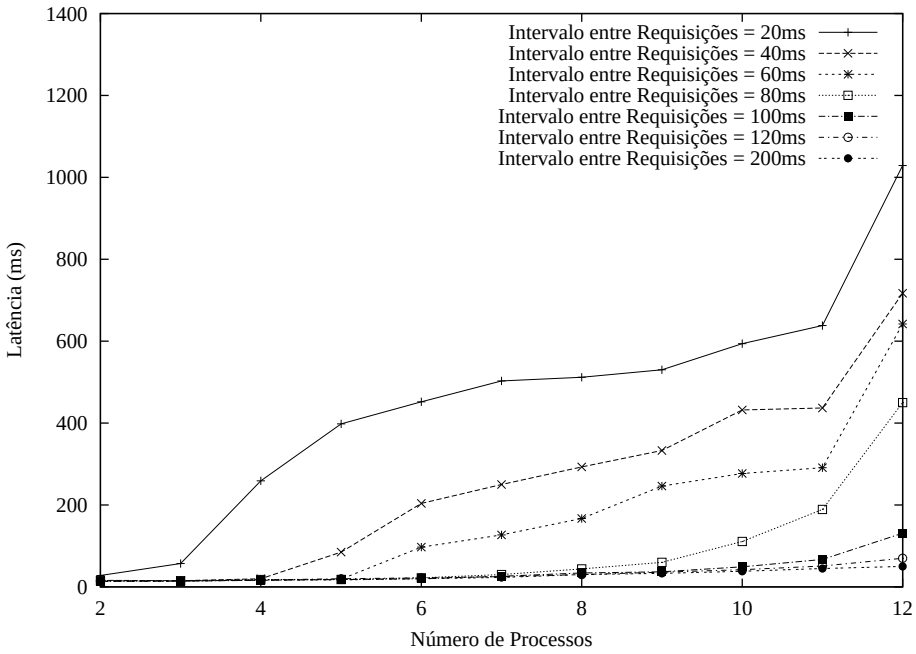

Figura 4: Tempo de entrega com variação no intervalo entre requisições

Cenário 3: Falha de Processos X Tamanho do Grupo Nesse cenário, analisamos o comportamento do protocolo na ocorrência de falhas de processos. Nesse caso, o protocolo de difusão atômica foi executado conjuntamente com um protocolo de *group membership* [8]. Esse protocolo é responsável pela instalação de uma nova visão de grupo na ocorrência de falhas de processos ou de solicitações de entrada ou de saída do grupo. Foram realizados cerca de mil testes; sendo que em cada um deles, o cliente emitiu 500 requisições a intervalos de $200ms$. Adotou-se um TC de $400ms$.

A figura 5 mostra os resultados obtidos através do seguinte experimento. No início da execução do protocolo, existem cinco (5) processos ativos no grupo. Por volta da requisição de número 60, ocorre a falha de um processo; os demais processos continuam a execução do protocolo. De acordo com o funcionamento do consenso, que se baseia no paradigma do coordenador rotativo, a cada quatro execuções, o processo falho será eleito como coordenador. Uma vez que esse processo já está falho, os demais precisarão esperar pela próxima rodada para iniciar um novo consenso. Esse comportamento é observado no gráfico pelo maior tempo de entrega a cada cinco consensos. Este valor está próximo dos $400ms$, o que corresponde exatamente ao valor de TC adotado. Essa situação se perpetua até que o processo falho seja removido do grupo pelo protocolo de *group membership* e uma nova visão seja instalada. Nesse momento, o tempo de entrega das requisições extrapola os $2000ms$, o que corresponde ao custo de realização de uma mudança de visão. Esse mesmo cenário se repete por volta da mensagem de número 250, quando um segundo processo falha. Esse teste é interessante para observarmos o quanto um procedimento de mudança de visão degrada o desempenho dos protocolos de acordo como um todo. Na prática, a mudança de visão deveria ser adiada pela aplicação enquanto fosse possível.

5.1. Avaliação e Comparação com Protocolos Similares

Encontramos na literatura a descrição de alguns resultados de desempenho para protocolos baseados no consenso e cujos experimentos foram realizados no mesmo contexto: rede local Ethernet a 10 Mbit/s e uso de máquinas Sun UltraSparc.

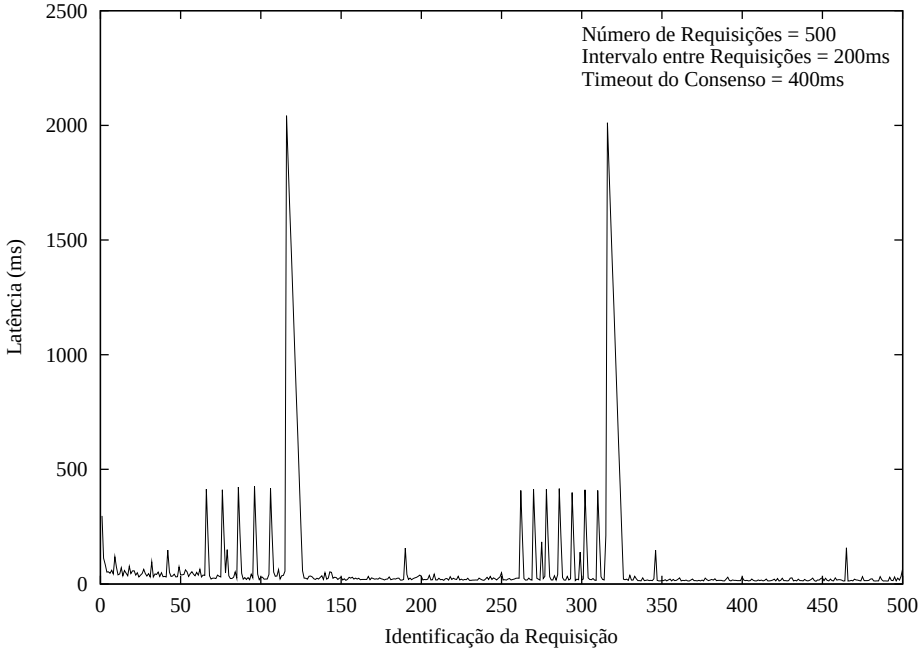

Figura 5: Tempo de entrega com ocorrência de duas falhas de processos

OGS [5] é um serviço de grupo cujo protocolo de difusão atômica é obtido a partir da instanciação, em tempo de execução, de parâmetros de um consenso genérico [10]. Diferentemente do nosso protocolo, OGS faz uso uma primitiva de difusão confiável e de canais confiáveis. Nos testes que realizou, ele adota o protocolo TCP/IP, configurado para transmissão imediata. Dos resultados exibidos nos seus gráficos, observamos que para grupos de até 7 processos, o desempenho do procotolo OGS é sempre superior ao nosso. Em compensação, a partir de 8 processos, a curva exibida pelo nosso protocolo cresce muito mais lentamente do que a curva do protocolo OGS. Para grupos de 10 processos por ex., nosso protocolo tem uma latência de pouco menos de $60ms$, enquanto OGS exibe uma latência de $80ms$. O protocolo proposto em [17] também faz uso de uma primitiva de difusão confiável, realiza seus testes numa rede TCP/IP e apresenta comportamento semelhante ao do OGS. Essas observações nos levam à seguinte conclusão: na prática, a complexidade inerente à primitiva de difusão confiável passa a ser significativa a partir de um determinada número de processos, e nesse caso, o nosso protocolo supera os demais. Quanto ao seu comportamento inferior para grupos de tamanhos pequenos; talvez, o que esteja interferindo no seu desempenho, seja o protocolo de consenso adotado.

Os resultados obtidos pelos nossos testes apresentam o desempenho do protocolo proposto quando os processos encontram-se numa rede local. O protocolo de consenso adotado tem por base a sincronização entre os relógios, o que torna o seu uso quase restrito a pequena escala. Para avaliarmos melhor o impacto da eliminação da primitiva de difusão confiável e do uso de canais não-confiáveis, precisaríamos testar a difusão atômica num contexto menos localizado, em que de fato, mensagens se perdem. Atualmente, estamos realizando uma implementação do protocolo GAF que se baseia no uso de detectores de falhas e que poderá ser utilizada num protocolo estendido em larga escala. Assim, essa análise mais apurada, poderá ser realizada.

6. Conclusão

Neste artigo, apresentamos um protocolo original de difusão atômica que, diferentemente dos demais protocolos similares até então propostos, apresenta mecanismos para lidar diretamente com a perda de mensagens da aplicação e implementa a entrega atômica das mensagens sem fazer uso da primitiva de difusão confiável. O protocolo foi implementado e testes experimentais foram realizados. Os resultados de desempenho obtidos pelos testes são encorajadores e recomendam o uso do nosso protocolo à medida que o tamanho do grupo cresçe. O algoritmo proposto está sendo utilizado na confecção e implementação do componente de replicação ativa da biblioteca de componentes de acordo ADAM [9].

Agradecimentos

Os autores gostariam de agradecer a valiosa contribuição de Michel Hurfin e de Frèdèric Tronel, pesquisadores da equipe *Adep* do IRISA-INRIA, França, a este trabalho.

Referências

[1] E. Anceaume. A lightweight solution to uniform atomic broadcast for asynchronous systems. In *Proceedings of The Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing (FTCS'97)*, pages 292–303, Washington - Brussels - Tokyo, June 1997. IEEE.

[2] A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG96)*, pages 105–122, 1996.

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

[4] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3:146–158, March 1989.

[5] P. Felber. *The CORBA Object Group Service: A Service Approach to Object Groups in CORBA*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1998.

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

[7] E. Fromentin, M. Raynal, and F. Tronel. A probabilistic analysis of the consensus problem. Technical Report 1226, IRISA, Rennes/France, January 1999. 23 pages.

[8] F. Greve, M. Hurfin, M. Raynal, and F. Tronel. Primary component asynchronous group membership as an instance of a generic agreement framework. In *ISADS'2001: 5th International Symposium on Autonomous Decentralized Systems*, pages 93–100, March 2001.

[9] F. G. P. Greve. *Réponses efficaces au besoin d'accord dans un groupe*. PhD thesis, IRISA - Université de Rennes I, France, November 2002.

[10] R. Guerraoui and A. Schiper. Consensus service: A modular approach for building fault-tolerant agreement protocols in distributed systems. In *Proceedings of the 26th International Symposium on Fault-Tolerant Computing (FTCS-26)*, pages 168–177, Sendai, Japan, June 1996.

[11] R. Guerraoui and A. Schiper. The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41, January 2001.

[12] V. Hadzilacos and S. Toueg. *Distributed Systems*, chapter Fault Tolerant Broadcasts and Related Problems, pages 97–145. Addison-Wesley, 1993.

[13] M. Hurfin, R. Macêdo, M. Raynal, and F. Tronel. A generic framework to solve agreement problems. In *Proc. of the* 19^{th} *IEEE Symposium on Reliable Distributed Systems (SRDS'99)*, pages 56–65, Lausanne, Switzerland, October 1999.

[14] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[15] A. Mostefaoui and M. Raynal. Low-cost consensus based atomic brodcast. In *Proceedings of IEEE Pacifi Rim Intern. Symposium on Dependable Computing (PRDC-00)*, Los Angeles, CA, December 2000. IEEE.

[16] F. Pedone and A. Schiper. Optimistic atomic broadcast. In *Proceedings of the 12th International Symposium on Distributed Computing (DISC'98, formerly WDAG)*, September 1998.

[17] F. Pedone and A. Schiper. Handling message semantics with generic broadcast protocols. *Distributed Computing*, 15(2):97–107, 2002.

[18] L. Rodrigues and P. Veríssimo. Topology-aware algorithms for large scale communication. *LNCS: Advances in Distributed Systems*, (1752):1217–1256, 2000.

[19] F. Schneider. *Distributed Systems*, chapter Replication Management using the State Machine Approach, pages 169–198. Addison-Wesley, 1993.

A. Prova da Correção do Protocolo de Difusão Atômica

Na demonstração que se segue, estamos assumindo a correção do protocolo GAF [13]. A demonstração das propriedades AB_Integridade e AB_Validade são triviais e ficam a cargo do leitor. Uma prova informal das demais propriedades é dada a seguir.

Teorema 1 AB_Ordem_Total*: se dois processos* p_i *e* p_j entregam *as mensagens* m *e* m'*, então* p_i *entrega* m *antes de* m'*, se e somente se,* p_j *entrega* m *antes de* m'*;*

Prova Um processo só entrega uma mensagem após ela ter sido ordenada (linhas 13-14). Pela propriedade C_Acordo_Uniforme do consenso, cada processo recebe o mesmo conjunto $Decided^k$ de mensagens ordenadas. Após a ordenação, as mensagens são armazenadas em $Undelivered_i$ (linhas 18-19), por ordem linear de consenso (k) e pela aplicação de uma função determinista sobre as suas identidades. A entrega das mensagens na linha 5 segue a ordem em que elas foram armazenadas em $Undelivered_i$. Assim, as mensagens são entregues na mesma ordem para todos os processos.

$\square_{Teorema\ 1}$

Teorema 2 AB_Acordo_Uniforme*: Se um processo* entrega *uma mensagem m então todos os demais processos corretos também entregam m;*

Prova Um processo só entrega mensagens que foram anteriormente ordenadas através do protocolo GAF. Pela propriedade de C_Acordo_Uniforme do consenso, todos os processos corretos terão acesso ao mesmo conjunto de mensagens ordenadas. Após a ordenação, estas mensagens serão incorporadas à fila $Undelivered_i$ (linhas 18-19), para serem posteriormente entregues a partir da execução da primitiva A_DELIVER() (linhas 4-6). Devido à possibilidade de perda de mensagens, alguns processos podem não ter recebido da rede estas mensagens; neste caso, após a ordenação, eles irão solicitá-las ao grupo (linhas 23-24). Sabe-se, pela definição da função $\mathcal{F}$ de GAF, que ao menos uma maioria de processos possui as mensagens. Além disso, supõe-se que uma maioria de processos seja correta, logo tem-se que ao menos um processo correto possuirá a mensagem que foi ordenada e poderá transmiti-la aqueles que não a possuem. Isso é realizado pelo protocolo das linhas 7-10. Desta maneira, todos os processos corretos terminarão por receber e entregar todas as mensagens ordenadas e o teorema segue. $\square_{Teorema\ 2}$

Teorema 3 AB_Terminação*: se um processo correto* envia *uma mensagem m então todos os demais processos corretos* entregam *m;*

Prova As mensagens são enviadas ao grupo de processos pela invocação de A-BROADCAST(). Toda mensagem m recebida através desta primitiva é armazenada em $Received_i$ (linhas 2-3). Posteriormente, ela será proposta ao consenso, através da função GET() (linhas 16-17). Pela definição dos parâmetros de GAF e pela propriedade C_Terminação do protocolo, uma decisão terminará por ser realizada. Os parâmetros de GAF que poderiam bloquear o consenso são GET() e ACCEPTABLE. Ocorre que, nos dois casos, faz-se uso de um mecanismo de timeout para que o controle possa ser retornado a GAF e o cáculo possa ter continuidade. Sabe-se, pela definição da função $\mathcal{F}$ de GAF, que nem todas as mensagens propostas ao consenso serão ordenadas. Se m não faz parte da decisão obtida, através do protocolo definido nas linhas 21-22, a cada insucesso, o processo fará a retransmissão de m até que ela venha a ser ordenada. Como assume-se a existência de canais com perdas eqüitativas, eventualmente m será recebida por uma maioria de processos corretos. Nesse caso, pela definição de $\mathcal{F}$, m fará parte de uma decisão e será armazenada no conjunto $Decided_i$ (linha 13). Posteriormente, pelas linhas 18 e 19, m fará parte da fila $Undelivered_i$. Em determinado momento, m estará no topo desta fila e pelas linhas 5 e 6, ela será entregue pelo processo i. Pelo Teorema 2, m será eventualmente entregue a todos os demais processos corretos e o teorema segue.

$\square_{Teorema\ 3}$