Difusão Atômica com Suporte à Perda de Mensagens*

Fabíola Gonçalves Pereira Greve

[1]LaSiD - Laboratório de Sistemas Distribuídos
Universidade Federal da Bahia
Campus de Ondina, 40170-110 Bahia, Brasil

fabiola@ufba.br

Resumo. *Apresentamos um protocolo de* difusão atômica *que suporta a perda de mensagens provenientes dos clientes e implementa diretamente a entrega atômica das mesmas sem recorrer ao uso de uma primitiva de* difusão confiável. *Ao nosso conhecimento, nenhum outro protocolo similar até então proposto apresenta tais mecanismos para lidar diretamente com as perdas. Este protocolo foi obtido a partir da especialização de um serviço genérico de consenso. Além disso, ele foi utilizado na implementação do componente de replicação ativa da biblioteca de componentes de acordo* ADAM *[1].*

Abstract. *We provide an efficient and realistic* atomic broadcast *protocol which supports the loss of messages from clients. As soon as we know, this is the only protocol proposed in the literature that deals directly with losses without using the* reliable broadcast *primitive as a resource to deliver messages in a reliable manner. This protocol was designed as a specialization of a general agreement framework. Moreover, it was used in the implementation of an active replication component that belongs to a library of agreement components called* ADAM *[1].*

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* [2] todas as réplicas têm o mesmo papel. Para preservar o estado coerente do servidor, uma primitiva de *difusão atômica* [3] 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*) [4]. 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 apresentado suporta a perda de mensagens provenientes dos clientes. Ao nosso conhecimento, nenhum outro protocolo similar até então proposto [5,

*Este trabalho foi realizado com financiamento parcial do CNPQ/Brasil (200.323-97).

6, 7, 8, 9, 10] fornece mecanismos para lidar diretamente com a possibilidade de tais perdas. Todos eles se fundamentam na existência de canais de comunicação confiáveis e/ou se baseiam no uso de uma primitiva de difusão confiável [3] para a entrega segura destas mensagens. A implementação de tal primitiva tem um alto custo: para cada mensagem proveniente do cliente, $O(n^2)$ mensagens são retransmitidas às n réplicas do servidor. Trabalhos foram propostos com o intuito de diminuir o custo da difusão confiável [11]. Nosso interesse, entretanto, é o de evitar o seu uso na implementação da difusão atômica.

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, descrevemos o protocolo de difusão atômica (seção 4.) e finalmente concluimos.

2. Modelo de Replicação Ativa

Designamos por S um servidor único particular. Para tolerar falhas definitivas (do inglês, *crash*) [12, 5], 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. A cardinalidade $|\Pi| = n$ é o grau de replicação do servidor. Os processos se comunicam e cooperam pela emissão de mensagens através de canais de comunicação segundo um modelo de comunicação assíncrono. 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* [13]. Finalmente, um *processo correto* é aquele que obedece à sua especificação e não falha durante toda a execução do sistema.

2.1. O Problema do Consenso

Informalmente o problema do consenso [12, 5] 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. Esse problema fundamental não tem solução determinista num sistema assíncrono, mesmo em presença de uma única falha [12]. 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* [5].

2.2. Detecção de Falhas

Informalmente, um detector de falhas não confiável é um conjunto de "oráculos"que fornece dicas aos processos sobre quais deles estão falhos. A cada processo p_i é associado um módulo de detecção que fornece informações de falhas através de uma lista, chamada $suspected_i$, contendo os processos suspeitos. Como em [5], consideramos que "p_i suspeita um processo p_j se $p_j \in suspected_i$ num instante t". Qualquer implementação de detector que satisfaça às exigências acima descritas pode ser usada. Um protocolo simples que emite mensagens periódicas do tipo "eu estou vivo", e utiliza um mecanismo de temporização para registrar as suspeitas, atende a essas exigências.

Chandra e Toueg [5] definem formalmente algumas classes de detectores. A classe $\Diamond\mathcal{S}$ (*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. O interesse da classe $\Diamond\mathcal{S}$ reside na sua importância para a resolução do consenso: ela representa a classe de detectores mais fraca

a permitir uma resolução do consenso e à condição que uma maioria de processos esteja correta ($f < n/2$) [14]. Os algoritmos aqui apresentados consideram o modelo assíncrono estendido com os detectores de falhas do tipo $\Diamond\mathcal{S}$; além disso, supõe-se que exista pelo menos uma maioria de processos corretos no sistema ($f < n/2$).

2.3. Difusão Atômica

Informalmente, um serviço de difusão atômica (*atomic broadcast*) assegura que os processos de um grupo de servidores entregarão um mesmo conjunto de mensagens enviadas por clientes e na mesma ordem total. Formalmente, esse serviço é definido da seguinte maneira [3, 5]:

- 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';

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

GAF (do inglês, G*eneral* A*greement* F*ramework*) [4] é um framework genérico para a realização de um acordo a partir do protocolo clássico de Chandra e Toueg [5]. O framework 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.

- 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.
- $\mathcal{F}$: 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 significativo.
- EXCUSED: predicado que autoriza um processo a não participar do consenso. Ele estabelece circunstâncias em que o valor proposto pelo processo é dispensável. Por exemplo, para resolver o problema do consenso propriamente dito, basta que uma maioria de valores seja coletada.

4. Difusão Atômica com Suporte à Perda de Mensagens

Grande parte dos protocolos de difusão atômica propostos a partir de uma redução a um serviço de consenso [5, 6, 7, 8] 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. Infelizmente, tal contexto não corresponde às características das redes e dos ambientes de computação atuais. Além disso, mesmo quando esses protocolos admitem a perda de mensagens [9, 10], eles costumam se apoiar no uso de uma primitiva de difusão confiável (*reliable broadcast*) [3] para propagar ao grupo de servidores as mensagens 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 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, propomos 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 provenientes dos clientes. Um mecanismo de retransmissão de mensagens adequado, 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.

Interface com a aplicação. Supomos que os clientes geram um fluxo contínuo de requisições através de alguma primitiva de difusão ao grupo de servidores. O processo servidor que recebe uma mensagem m do cliente chama o procedimento A_BROADCAST(m) para divulgar m de maneira atômica a todos os demais membros do grupo. Os processos entregam as mensagens localmente por intermédio do procedimento A_DELIVER(), chamado pela camada da aplicação responsável pela execução das requisições.

Princípio. A ordenação das mensagens emitidas pelos clientes, e disseminadas a partir da primitiva A-BROADCAST(), 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, a partir da operação 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 se ocupa da 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 em $Received_i$ (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, 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). 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).

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.

Atomic Broadcast
(1) $Received_i \leftarrow \emptyset$; $A\_Delivered_i \leftarrow \emptyset$; $Undelivered_i \leftarrow \emptyset$; $k \leftarrow 0$;
cobegin
thread AB1:
% Interface com a aplicação %
(2) **upon** a call to A_BROADCAST(m) **do**
(3) $Received_i \leftarrow Received_i \cup \{m\}$; **enddo**
(4) **upon** a call to A_DELIVER() **do**
(5) **wait until** ($Undelivered_i \neq \emptyset$); $m \leftarrow$ remove_first($Undelivered_i$);
(6) **wait until** ($m \in Received_i$); *return* (m); **enddo**
% Interface com a camada de comunicação %
(7) **upon** reception of REQUEST_MSG (p_j, m) **do**
(8) **if** ($m \in Received_i$) **then** *send* A_MSG (m) to p_j; **endif enddo**
(9) **upon** reception of A_MSG(m) **do**
(10) $Received_i \leftarrow Received_i \cup \{m\}$; **enddo**

thread AB2: % Cálculo do conjunto de mensagens e da sua ordem de entrega %
(11) **while** ($true$) **do**
(12) $k \leftarrow k + 1$;
(13) $Decided^k \leftarrow$ GAF();
(14) Entrega_Confiável ($Decided^k$);
(15) **enddo**
coend

Function GET() % Determina um valor de proposição significativa %
(16) **wait until** ($Received_i \setminus A\_Delivered_i \neq \emptyset$) *or* (*expired timeout*);
(17) $Proposed^k \leftarrow Received_i \setminus A\_Delivered_i$; $return(Proposed^k)$;

Procedure Entrega_Confiável($Decided^k$)
(18) $Ordered^k \leftarrow Decided^k \setminus A\_Delivered_i$;
(19) Queue ($Undelivered_i, Ordered^k$) % coloca mensagens na fila do conjunto %
(20) $A\_Delivered_i \leftarrow A\_Delivered_i \cup Ordered^k$;
(21) **for each** ($m \in Proposed^k \setminus Decided^k$) **then**
(22) *broadcast*(m); **endif** % difunde as mensagens recebidas mas ainda não ordenadas %
(23) **for each** ($m \in Decided^k \setminus Received_i$) **then**
(24) *broadcast* REQUEST_MSG(p_i, m); **endif** % solicita mensagem que falta %

Figura 1: Difusão Atômica com Perda de Mensagens

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 AB_Acordo_Uniforme e AB_Terminação definidas para o problema. 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 entre 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 ACCEPTABLE(v) – Ela retorna $verdadeiro$ quando o valor v não é o conjunto vazio $\emptyset$. 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 possa ser tomada.

• 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). 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 é 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.

Parâmetro	Descrição
GET	Retorna ($Received_i \setminus A\_Delivered_i$) ou (*expiração de timeout*)
$\mathcal{F}$	Retorna intersecção de todos os conjuntos de entrada: $\bigcap Proposed_i$
ACCEPTABLE	Retorna $verdadeiro$ quando não é aplicado sob um subconjunto vazio
EXCUSED	EXCUSED(p_i) retorna $verdadeiro$ após a expiração de um certo tempo (timeout)

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

Estabilização das Mensagens. O procedimento de estabilização de mensagens atende a dois objetivos: i) retransmissão/recuperação de mensagens perdidas e ii) eliminação local de mensagens estáveis, i.e., 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. As mensagens ordenadas são entregues à aplicação sem necessidade de esperar que elas sejam estáveis no grupo. Tem-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. O procedimento de estabilização utiliza então 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-lo aos outros membros (no caso

da mensagem não ter sido ordenada). Para as mensagens que já foram ordenadas, elas só serão re-enviadas aos processos que não as receberam da rede.

5. Conclusão

Apresentamos um protocolo original de difusão atômica que, diferentemente dos demais protocolos similares até então propostos, considera a possibilidade de perda de mensagens dos clientes e implementa diretamente a entrega atômica das mensagens sem fazer uso da primitiva de difusão confiável. O algoritmo obtido foi utilizado na confecção e implementação do componente de replicação ativa da biblioteca de componentes de acordo ADAM [1]. Este componente encontram-se atualmente em fase de testes e de avaliação de desempenho. Pretendemos, através desta análise quantitativa, corroborar nossas afirmações com relação à eficiência do protocolo proposto.

Agradecimentos

A autora gostaria de agradecer a contribuição de Michel Hurfin e de Frederic Tronel, pesquisadores da equipe *Adep* do IRISA-INRIA, França, a este trabalho.

Referências

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

[2] F. Schneider, *Distributed Systems*, ch. Replication Management using the State Machine Approach, pp. 169–198. Addison-Wesley, 1993.

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

[4] 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)*, (Lausanne, Switzerland), pp. 56–65, Oct. 1999.

[5] T. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of ACM*, vol. 43, pp. 225–267, Mar. 1996.

[6] 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)*, (Washington - Brussels - Tokyo), pp. 292–303, IEEE, June 1997.

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

[8] 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), IEEE, Dec. 2000.

[9] 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)*, (Sendai, Japan), pp. 168–177, June 1996.

[10] R. Guerraoui and A. Schiper, "The generic consensus service," *IEEE Transactions on Software Engineering*, vol. 27, pp. 29–41, Jan. 2001.

[11] L. Rodrigues and P. Veríssimo, "Topology-aware algorithms for large scale communication," *LNCS: Advances in Distributed Systems*, no. 1752, pp. 1217–1256, 2000.

[12] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of ACM*, vol. 32, pp. 374–382, Apr. 1985.

[13] 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)*, pp. 105–122, 1996.

[14] T. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," *Journal of ACM*, vol. 43, pp. 685–722, July 1996.

A Prova Informal da Correção do Protocolo de Difusão Atômica

Na demonstração que se segue, estamos assumindo a correção do protocolo GAF [4]. A demonstração das propriedades de AB_Integridade e AB_Validade são triviais e ficam a cargo do leitor. A propriedade AB_Ordem_Total é satisfeita porque i) as mensagens são entregues por ordem linear de consenso e pela aplicação de uma função determinista sobre as suas identidades ii) pela propriedade *Acordo_Uniforme* do consenso, cada processo recebe o mesmo conjunto de mensagens ordenadas. Uma prova informal das demais propriedades é dada a seguir.

Teorema 1 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 *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. $Teorema\ 1$

Teorema 2 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 da primitiva 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). Sabe-se, pela definição da função $\mathcal{F}$ de GAF, que nem todas as mensagens propostas ao consenso serão ordenadas. Nesse caso, através do protocolo definido nas linhas 21-22, 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 será ordenada; pelo Teorema 1, m será eventualmente entregue a todos os processos corretos.

$Teorema\ 2$