Um Protocolo de Validação Atômica Não-Bloqueante Eficiente*

Fabíola Greve⋆† Jean-Pierre Le Narzul•†
{fgreve|jlenarzu}@irisa.fr

⋆ LaSiD/DCC-UFBA
Campus de Ondina
40170-110 Bahia, Brasil

† IRISA/INRIA
Campus de Beaulieu
35042 Rennes, France

• ENST-Bretagne
Campus de Rennes
35512 Cesson-Sévigné, France

Resumo

Apresentamos um protocolo original não-bloqueante de *Validação Atômica* baseado no *Consensus* que é mais eficiente que todos os protocolos equivalentes propostos até então. Em presença de um baixo número de falhas toleradas pelo sistema ($f \leq 2$) ele difunde quase o mesmo número de mensagens que o protocolo clássico 2PC (*Two-Phase Commit*). No caso genérico, ($f < n/2$), exibe uma complexidade de $n^2 + f(n)$, enquanto que todas as outras soluções apresentam uma complexidade de $2n^2 + f(n)$ mensagens ponto a ponto. Este protocolo é o resultado da aplicação de uma nova metodologia de desenvolvimento para resolver *problemas de acordo* que leva em conta informações semânticas. Ele é baseado no uso de um protocolo de *consensus* [1] recentemente proposto que permite decisões em um único passo de comunicação em condições favoráveis.

Abstract

We present an original non-blocking *Consensus*-based *Atomic Commitment* algorithm that performs better than other existing equivalent protocols. In presence of a low resiliency rate (number f of process crashes tolerated), it broadcasts almost the same number of messages as the classical 2PC (*Two-Phase Commit*) protocol. In the general case, ($f < n/2$), it exhibits a complexity of $n^2 + f(n)$ while all the other solutions presents a complexity of $2n^2 + f(n)$ point to point messages. This protocol is the result of the application of a new development methodology to solve agreement problems that takes into account information about the semantics of the particular problem. It is based on the use of recent *consensus* protocols [1] that allows decisions in one-step communication in good circumstances.

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

1 Introdução

O problema da *validação atômica* (do inglês, *atomic commitment)* é essencial na implementação de bases de dados distribuídas e monitores transacionais, sendo responsável pela propriedade de *atomicidade em presença de falhas* do modelo de transações. Esta propriedade estabelece uma tomada de decisão uniforme (validar ou não a transação) da parte de todos os participantes corretos, mesmo na ocorrência de falhas.

A *validação atômica* pertence à classe de *problemas de acordo*. Tal classe incorpora diversos problemas práticos presentes na concepção de sistemas tolerantes a faltas, cujo denominador comum é o problema do *consensus*. Este envolve um conjunto de processos cujo objetivo é concordar num valor único. Importantes problemas de acordo (*difusão atômica* [2], *validação atômica* [5], *group membership* [6], *view synchrony* [7], etc.) podem ser vistos como uma redução ao *consensus*. Este problema essencial é assim considerado como um serviço de base em cima do qual outros protocolos podem ser construídos, visando a implementação de aplicações distribuídas confiáveis [7]. As soluções baseadas no *consensos* 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 clássico de *validação atômica em duas fases* (em inglês, *two-phase commit* - 2PC) [8] é a melhor solução até hoje proposta. Apenas *três* passos de comunicação[1] e $3n$ mensagens ponto a ponto são necessários para validar a transação. Infelizmente, em presença de falhas, ela é bloqueante. Um protocolo de acordo é considerado não-bloqueante quando ele permite a tomada de decisão pelos processos corretos mesmo na ocorrência de falhas. Os protocolos de *validação atômica em três fases* (em inglês, *three-phase commit* - 3PC) [9, 10] são não bloqueantes. Entretanto, tais protocolos têm um alto custo em presença de falhas, exigem condições fortes de sincronização, além de uma alta latência para terminar: *cinco* passos de comunicação e a difusão de $5n$ mensagens. Protocolos que consideram um serviço de *consensus* como protocolo de terminação para lidar com falhas foram sugeridos [11, 12, 13, 14]. Dentre estes, o trabalho proposto por [12] é o mais eficiente. Ele executa em *três* passos de comunicação e requer a emissão de ($4nf + 3n$) mensagens ponto a ponto (ou $n + 2f + 2$ mensagens de difusão (*broadcast*)).

Neste artigo, apresentamos um protocolo de *validação atômica* original, que é mais eficiente que todos os outros protocolos baseados no *consensus* propostos até então. Ele termina em *três* passos comunicação e requer a emissão de ($2nf + 3n$) mensagens ponto a ponto (ou $n + f + 2$ mensagens de difusão). Tal como em [12], o número de mensagens emitidas por este protocolo é proporcional ao número máximo de processos que podem falhar (f). Nenhum outro protocolo baseado no *consensus* exibe tal desempenho. Ele é o resultado da aplicação de uma nova metodologia de desenvolvimento para a obtenção de soluções a protocolos de acordo. Tal metodologia leva em consideração informações semânticas ligadas aos problemas e se baseia no uso de recentes protocolos de *consensus* [1] propostos pelos autores e que permitem decisões antecipadas em presença de condições favoráveis.

Este artigo está organizado da seguinte maneira. A Seção 2 apresenta o modelo de computação distribuída. Em seguida, na Seção 3, apresentamos em detalhes o problema da *validação atômica*, sua relação com o problema do *consensus* e as principais soluções existentes. Antes de apresentarmos o protocolo proposto (Seção 5), uma descrição do algoritmo de *consensus* no qual ele se baseia é realizada na Seção 4. Finalmente, a Seção 6 apresenta uma análise de

[1] Um passo de comunicação se caracteriza pela emissão e respectiva recepção de um conjunto de mensagens.

desempenho do protocolo e a sua comparação com as principais soluções existentes.

2 Modelo de Sistema Distribuído

2.1 Sistema Assíncrono

O modelo de sistema segue o padrão definido em [4, 2]. Ele consiste em um número finito Π de $n > 1$ processos, $\Pi = \{p_1, ..., p_n\}$ que se comunicam e cooperam pela emissão de mensagens através de canais de comunicação. Tal multiplicidade de processos, que interagem através de canais, faz com que o sistema seja considerado *distribuído*. 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. Em outras palavras, o *modelo* de sincronização é *assíncrono*. Sistemas abertos e em larga escala são tipicamente assíncronos.

Os *canais* são considerados *confiáveis*, na medida em que eles não criam, não alteram, não duplicam nem perdem as mensagens que ali trafegam.

Os *processos falham de maneira definitiva*. Um processo se comporta corretamente (de acordo com a sua especificação) até que venha a falhar. Nesse caso, ele para de executar o seu código respectivo e não se recupera. Em inglês, tal modo de falhas é dito *crash*. Por definição, um *processo correto* é aquele que não falha durante a execução do sistema. Um *processo faltoso* é aquele que não é correto.

2.2 O Problema do Consensus

Informalmente o problema do *consensus* 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 *consensus* é definido pelo seguinte conjunto de propriedades [4, 2].

- 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 é de extrema importância para a obtenção de soluções para outros problemas onde a necessidade de acordo é recorrente. É o caso do problema da *validação atômica*. Ocorre, entretanto, que o *consensus* não tem solução determinística num sistema assíncrono, mesmo em presença de uma única falha [4]. Isso acontece porque, devido às incertezas criadas pelo ambiente assíncrono e pela existência de falhas, não é possível distinguir com precisão entre um processo correto e um processo faltoso.

Uma das estratégias adotadas para contornar tal resultado de impossibilidade consiste em estender o modelo assíncrono com algum grau de "sincronia". Com este intuito, um dos avanços mais significativos é a proposta de uso dos *detectores de falhas* investigada por Chandra e Toueg [2].

2.3 Detecção de Falhas

Informalmente, um *detector de falhas* é um conjunto de *oráculos* que fornece dicas aos processos sobre quais deles estão falhos. O detector é não confiável porque ele pode se equivocar. Ou seja, tanto suspeitar da falha de processos corretos, quanto considerar que processos faltosos são corretos. Além disso, a detecção é não estável, o que significa que o detector pode mudar de opinião. A cada processo p_i é associado um módulo de detecção, que fornece informações de falhas através de uma lista de processos, chamada $suspected_i$, contendo os processos suspeitos. Como em [2], 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 *timeout* para registrar as suspeitas, atende a essas exigências.

Formalmente, um detector de falhas é definido por duas propriedades abstratas: a *completude* (do inglês, *completeness*) e a *corretude* (do inglês, *accuracy*). A *completude* ocupa-se da real detecção de falhas e a *corretude* limita os erros que podem ser cometidos pelo detector. A combinação dessas duas propriedades dá origem a algumas classes de detectores. Num dos extremos tem-se as classes que nunca cometem erros de detecção (possuem *corretude forte*) e são consideradas *confiáveis*; no outro extremo tem-se todas as classes que podem se equivocar e são ditas *não-confiáveis*.

A classe de *detectores de falhas forte definitiva*, também conhecida por $\diamond\mathcal{S}$, garante as seguintes propriedades: i) completude forte: todo processo falho será considerado suspeito por todo processo correto de maneira definitiva; e a ii) corretude fraca definitiva: existe un instante de tempo a partir do qual algum processo correto não será considerado suspeito por nenhum outro processo correto. O interesse do $\diamond\mathcal{S}$ reside na sua importância para a resolução do *consensus*: ele representa a classe de detectores mais fraca a permitir uma resolução do *consensus*, desde que uma maioria de processos esteja correta ($f < n/2$) [3]. O nosso trabalho considera portanto o modelo assíncrono estendido com os detectores de falhas do tipo $\diamond\mathcal{S}$.

3 O Problema da Validação Atômica Não-Bloqueante

O Conceito de *transação* assegura um tratamento consistente na manipulação de dados de um sistema de banco de dados distribuídos. O Problema da *validação atômica não-bloqueante* (do inglês, *non-blocking atomic commitment* - NBAC) está no centro de tal paradigma, e garante a consistência dos dados em presença de falhas de processos e dos canais de comunicação. Ele provê a propriedade de *atomicidade na presença de falhas* (também designada *tudo ou nada*), que assegura uma tomada de decisão uniforme, com relação às modificações efetuadas sobre os dados, da parte de todos os participantes da transação.

O resultado de uma transação depende das condições locais de cada participante para tornar permanentes as modificações efetuadas sobre os dados. Devido à ocorrência de certas dificuldades, tais como, as situações de blocagem (*deadlock*), de conflito de concorrência, esgotamento de uso da memória, etc., pode acontecer que alguns processos não estejam prontos a ratificar tais modificações. Um protocolo de validação atômica é executado ao fim da transação a fim de coletar a intenção de cada participante de validar (voto = *sim*) ou anular (voto = *não*) as modificações realizadas sobre os dados. O resultado do protocolo depende dos votos coletados: se tudo corre bem, a transação é validada (em inglês, COMMIT); se alguma coisa anda mal, ela é abandonada (em inglês, ABORT). Normalmente, se um número suficiente de processos vota *sim* (geralmente, todos), a decisão é COMMIT. Senão, se um bom número de processos vota *não*

(geralmente, pelo menos um), a decisão é ABORT.

Em resumo, a depender dos votos coletados dos participantes de uma transação distribuída e das condições de falhas do ambiente, o resultado de um protocolo de NBAC será COMMIT ou ABORT. De maneira formal, este problema é definido pelas seguintes propriedades [5] :

- AC_Terminação: todos os processos corretos decidem de maneira definitiva;
- AC_Integridade: um processo decide no máximo uma vez;
- AC_Acordo_Uniforme: dois processos não decidem diferentemente;
- AC_Validade: se um processo decide COMMIT, então todos os processos votaram *sim*.

  Para evitar decisões inesperadas, onde o valor decidido independe dos votos coletados, considera-se também uma propriedade de não trivialidade.

- AC_Não_Trivialidade: se todos os participantes votam *sim* e não existem *falhas* então a decisão é COMMIT.

3.1 Um Panorama das Soluções Existentes para o NBAC

O algoritmo clássico de *validação atômica em duas fases* (em inglês, *two-phase commit*), também conhecido por 2PC [8], é a melhor solução existente. Infelizmente, na ocorrência de falhas, ela é bloqueante e não satisfaz a propriedade de AC_Terminação. Isso acontece porque o controle da decisão é centralizado por um processo coordenador. A Figura 1 ilustra os passos de comunicação realizados por este protocolo. O coordenador solicita e espera os votos dos outros participantes da transação. Se todos votam *sim*, então este difunde a decisão COMMIT; se algum dentre eles vota *não* ou se existe alguma suspeita de falhas, o coordenador difunde ABORT. É fácil verificar que se o coordenador falha antes da difusão da decisão, os outros participantes da transação ficam bloqueados à espera de uma decisão do coordenador.

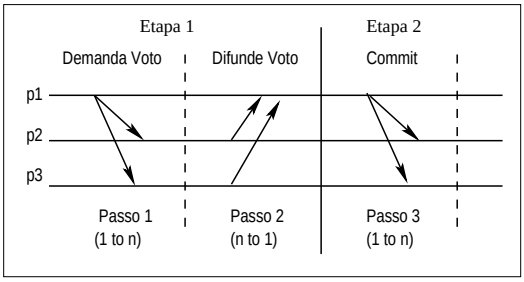

Figura 1: O protocolo 2PC (*two-phase commit*), com *coordenador* $= p_1$

Na prática, a possibilidade de ter decisões, mesmo na ocorrência de falhas, é uma característica muito desejável, pois independentemente do valor decidido, os processos que são corretos podem liberar ao sistema os recursos adquiridos ao longo da transação. Os protocolos de *validação atômica em três fases* (do inglês, *three phase commit*) (3PC) [9, 10] são não bloqueantes. Entretanto, as soluções apresentadas para evitar a blocagem são de difícil compreensão, prova, realização e teste [13]. Isso acontece porque o algoritmo adotado pelo 3PC centraliza igualmente a decisão sobre o processo coordenador. Para garantir um comportamento coerente na presença de falhas, um procedimento de recuperação é efetuado a fim de eleger um

novo coordenador. Durante tal processo de eleição, múltiplos coordenadores e múltiplas visões do grupo podem coexistir. Assim, para tomar uma decisão, o coordenador espera até que possa estabelecer um *quorum* (que normalmente consiste numa maioria de processos) seguro de participantes. Além do alto custo ligado ao procedimento de recuperação de falhas, tais protocolos precisam de detectores de falhas confiáveis para decidir.

Guerraoui [5] estudou a possibilidade de solucionar o problema de NBAC para o modelo assíncrono estendido com *detectores de falhas não confiáveis* [2]. Ele mostrou que o NBAC é mais difícil de resolver que o problema do *consensus* e, além disso, que diferentemente do *consensus*, ele é impossível de resolver com detectores de falhas não confiáveis. Tal resultado deve-se ao fato de que com tais tipos de detectores, as falhas não podem ser percebidas de maneira segura e dessa forma a propriedade de AC_Não_Trivialidade apresentada para o problema não poderá ser satisfeita. Ao definir uma nova propriedade com exigências mais fracas, um novo problema chamado *validação atômica fraca não bloqueante* (em inglês, *non-blocking weak atomic commitment* - NB-WAC) foi definido.

3.2 O Problema de Validação Atômica Fraca Não Bloqueante (NB-WAC)

Esse problema mantém todas as propriedades do NBAC, à exceção da *Não_Trivialidade*, que é enfraquecida para levar em consideração as *suspeitas* ao invés das *falhas* propriamente ditas.

- AC_Não_Trivialidade: se todos os participantes votam *sim* e não existem *suspeitas*, então a decisão é COMMIT.

Uma característica importante do NB-WAC é a sua redutibilidade ao *consensus* [5]. Assim, os resultados concernentes à solvabilidade do *consensus* em um modelo assíncrono com detectores de falhas não-confiáveis são também aplicados à resolução do NB-WAC.

Alguns protocolos de *validação atômica* foram obtidos a partir de uma redução ao *consensus* [11, 12, 13, 14]. As principais vantagens da utilização de tal abordagem são por um lado a precisa caracterização das condições de terminação do protocolo (desde que uma maioria se comunique) e, por outro lado, a maneira de concepção modular (uso do *consensus* como *framework* de base para assegurar tal terminação).

Além do seu caráter versátil, os protocolos baseados no uso do *consensus* são tão eficazes em termos de latência de comunicação quanto o clássico 2PC. Três passos de comunicação são requeridos para decidir em ausência de falhas. Entretanto, apesar da baixa latência, o número de mensagens difundidas com o objetivo de tolerar faltas ainda é importante: $O(n^2)$ mensagens ponto-a-ponto são necessárias, em oposição a somente $O(n)$ mensagens requeridas pelo 2PC. Segundo nosso conhecimento, o protocolo proposto por [12] é o que apresenta os melhores resultados: $O(nf)$ mensagens.

Neste artigo, apresentamos um protocolo original de complexidade $O(nf)$ de melhor desempenho do que [12]; em presença de um baixo valor de f, exibe um custo proporcional ao do 2PC. Ele foi obtido a partir de uma redução a um algoritmo de *consensus* [1] recentemente proposto. Este algoritmo será descrito na Seção 4 antes da apresentação do protocolo propriamente dito na Seção 5. Para efeito de uniformização, os algoritmos serão apresentadas num pseudocódigo em inglês.

4 Consensus em Um Único Passo de Comunicação

Em [1] apresentamos alguns protocolos de *consensus* que decidem em um único passo de comunicação em condições favoráveis. A estratégia sugerida consiste em enriquecer os processos com um "acordo a priori". Um acordo possível consiste em identificar um subconjunto de processos, conhecido de todos os demais. Considere a existência de um sub-conjunto $S \in \Pi$ de processos, cuja composição é conhecida de antemão por todos os participantes. Se todos os processos de S propõem o mesmo valor, então é possível decidir em um único passo de comunicação. O protocolo apresentado na Figura 2 utiliza essa filosofia. Ele utiliza um detector de falhas do tipo $\diamond\mathcal{S}$, requer $|S| > f$ e $f < n/2$. Na prática, o conjunto S deve ser escolhido de maneira a reunir os servidores mais confiáveis do sistema, já que eles serão responsáveis pela terminação antecipada do cálculo.

Princípio Realiza-se numa primeira fase uma tentativa de decisão antecipada; os participantes do consensus trocam os votos entre si, se um processo coleta todas as proposições provenientes dos processos em S e, além disso, elas contêm o mesmo valor, então ele pode decidir imediatamente. Os processos que não reuniram tal condição de término antecipado, iniciam a execução de um protocolo de *consensus* subjacente (Underlying_Consensus). Qualquer um dos *consensus* propostos na literatura podem ser usados como módulo subjacente [2].

Function Set_Consensus(v_i, S)
Task $T1$: % ——————- Phase 1: Early Deciding
(1) **if** ($p_i \in S$)
(2) *Broadcast* PROPOSE(v_i); **endif**
(3) **wait until** ($\forall\ p_j \in \Pi : p_j \in suspected\_i \vee$ PROPOSE message received from p_j)
and ($\exists\ p_j \in S$: PROPOSE message received from p_j);
(4) **if** (the same value v has been received from each process $\in S$) **then**
(5) *Broadcast* C_DECISION(v); *return*(v);
% ——————- Phase 2: Deciding by Consensus
(6) **else** $v_i \leftarrow$ a value from a process $\in S$;
(7) *return*(Underlying_Consensus(v_i));
(8) **endif**
Task $T2$: (9) **upon** *reception* **of** C_DECISION(v): *Broadcast* C_DECISION(v); *return*(v)

Figura 2: Módulo de Consensus

Funcionamento Cada processo participante do *consensus* chama a função Set_Consensus(v_i, S) passando o seu valor de proposição v_i e o conjunto S. A função termina pela execução das linhas 5, 7 ou 9 e o retorno do valor decidido. Para evitar que um processo fique bloqueado para sempre, à espera de um valor decidido, sempre que um processo decidir, ele difunde o valor de decisão através de uma primitiva de *difusão confiável* [2]. Tal primitiva garante a recepção de uma mensagem por todos os processos que são corretos. A função é assim composta de duas tarefas concorrentes $T2$: responsável pela *difusão confiável*, e $T1$: que implementa o protocolo propriamente dito.

Na primeira fase de $T1$ somente os processos pertencentes a S participam da emissão do valor inicial (linhas 1-2); todos os demais processos esperam os valores de proposição provenientes dos processos desse subconjunto (linha 3). Quando um processo p_i verifica que um mesmo valor v foi proposto por *todos* os processos pertencentes a S, ele pode decidir v de

maneira segura e em um único passo de comunicação (linhas 4-5). Isto é possível porque, dado que existem ao menos $(f + 1)$ processos em S, todo processo correto recebe ao menos (1) mensagem PROPOSE proveniente de um processo em S. Consequentemente, esses processos podem adotar v como valor de proposição na linha 6. Quando todos os processos de S propõem o mesmo valor v, aqueles processos que não decidem na linha 5, invocam Underlying_Consensus na linha 7 com o mesmo valor v. Assim, v é necessariamente o valor decidido.

5 NB-2PC: Um Protocolo Não-Bloqueante de Validação Atômica Fraca

A Figura 3 apresenta um diagrama hierárquico das funções envolvidas na implementação do protocolo de validação atômica. Ele será designado NB-2PC. Uma transação (Transaction) envolve uma função de validação atômica (Atomic_Commit()) que por sua vez é implementada através do protocolo de consensus (Set_Consensus()). Cada um desses módulos consulta a lista $suspected_i$ do detector de falhas ligado ao processo.

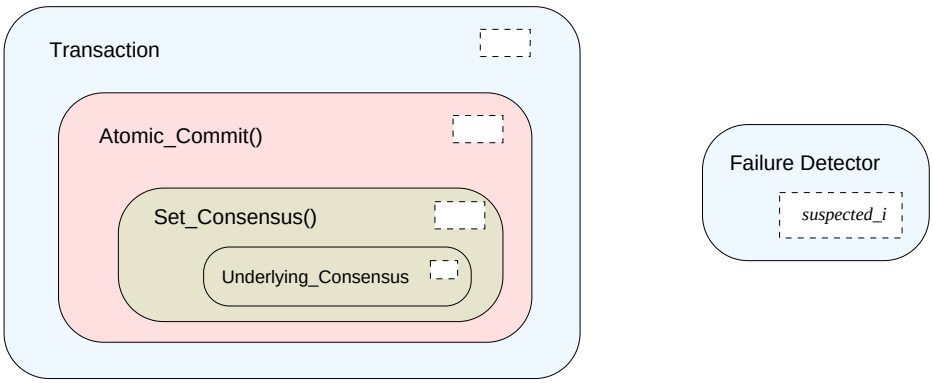

Figura 3: Diagrama Hierárquico Entre as Diversas Funções do Protocolo NB-2PC

Procedure Transaction
(1) **if** $(p_i = leader)$ **then** % The leader starts the transaction %
(2) *Broadcast* REQUEST_VOTE<>; **endif**
(3) **wait until** (REQUEST_VOTE<> received from $leader$) $\vee$ ($leader \in suspected_i$);
(4) **if** (REQUEST_VOTE<> received from $leader$) **then**
(5) **if** (able to make updates permanent)
(6) **then** $vote_i \leftarrow$ SIM;
(7) **else** $vote_i \leftarrow$ NÃO; **endif**
(8) **else** $vote_i \leftarrow$ NÃO; **endif**
(9) $return$ Atomic_Commit($vote_i$);

Figura 4: Módulo de Transação

Módulo de Transação A Figura 4 mostra o módulo responsável pela implementação de uma transação. Ele é executado por cada um dos processos participantes e se inicia por um desses processos, chamado líder. O líder convida todos os participantes a declararem sua intenção

de validar as operações sobre os dados (linhas 1-2). Os processos votam *sim* (linha 6) se eles estão prontos a validar as modificações; caso contrário, se eles não estão ainda prontos ou se suspeitam do líder, eles votam *não* (linhas 7-8). Depois de preparar o seu voto local, cada participante invoca a função Atomic_Commit() (linha 9). Essa função implementa um protocolo de *validação atômica fraca não-bloqueante* e assegura a obtenção de um resultado único e uniforme para a transação, independentemente da ocorrência de falhas.

```
Function Atomic_Commit(vote_i)
cobegin
[Task T1] % ———————— Phase 1: Exchange de Votes
(1)  SEND VOTE< vote_i > to all p_j : p_j ∈ S ;   % |S| > f, S ⊆ Π %
(2)  if (vote_i = NÃO) then Broadcast AC_DECISION < ABORT>;  return(ABORT); endif
(3)  if (p_i ∈ S) then
(4)     wait until (vote_j received from all p_j ∈ (Π − suspected_i));
% ———————— Phase 2: Exchange de propositions
(5)     if (∀ p_j ∈ Π : (vote_j received) ∧ (vote_j = SIM))
(6)        then return Set_Consensus (COMMIT, S);
(7)        else return Set_Consensus (ABORT, S);  endif
(8)  else  return Set_Consensus (⊥, S);  endif
[Task T2]
(9)  upon (reception AC_DECISION < v >) Broadcast AC_DECISION < v >;  return(v);
coend
```

Figura 5: Módulo de Validação Atômica

Princípio A Figura 5 apresenta um proposição para a função Atomic_Commit(). Tal função faz uso das possibilidades de terminação antecipada introduzidas pelo algoritmo Set_Consensus() apresentado na Seção anterior. O princípio de base é o seguinte: Numa *primeira fase*, os processos difundem e coletam sua disposição de tornar as modificações sobre os dados permanentes (votam *sim* ou *não*). Com base nos votos coletados, eles iniciam uma *segunda fase* em que difundem a sua proposição de validação (COMMIT) ou anulação (ABORT) da transação. Tal proposição é realizada através do serviço de *consensus* (Set_Consensus()). O algoritmo pode acabar no primeiro passo do *consensus* caso os processos reunam as "condições favoráveis de terminação" – todos os processos em S votam igual: eles optam ou pelo COMMIT ou pelo ABORT.

Funcionamento Se um processo p_i ainda não está pronto para homologar as modificações ($vote_i = \textit{não}$) ele decide imediatamente e de maneira isolada por anular a transação (ABORT), difundindo de maneira confiável sua decisão (linha 2). Essa decisão é legítima, pois ABORT é a única resolução aceitável a fim de respeitar as propriedades definidas para o problema. A *difusão confiável* é necessária para assegurar o acordo e a terminação do cálculo. A tarefa concorrente $T2$ (linha 9) implementa tal primitiva.

O participante que está pronto a validar a transação ($vote_i = \textit{sim}$) inicia o protocolo a partir de uma difusão seletiva somente aos membros do conjunto S. Na *primeira fase*, todos os participantes difundem aos processos de S os seus respectivos votos (linha 1), e estes esperam a recepção das mensagens de todos aqueles que não são suspeitos (linhas 3-4). Na *segunda fase*, somente os processos de S propõem ao *consensus* COMMIT ou ABORT de acordo com os votos coletados (linhas 5-7). Os outros processos ($\Pi - S$) não participam da difusão da estimativa inicial em Set_Consensus(v_i) (linha 1, da Figura 2) e podem então chamar esta

função passando como parâmetro inicial um valor não significativo ($v = \perp$)(linha 8). Se todos os processos pertencentes a S votam da mesma maneira (seja por COMMIT, seja por ABORT), então é possível decidir desde a primeira fase de Set_Consensus() (linhas 1-5, da Figura 2). Aqueles processos que não conseguiram decidir antecipadamente, iniciam numa *terceira fase*, o algoritmo Underlying_Consensus para a obtenção de um resultado uniforme para a transação. Como iremos ilustrar na próxima Seção, a difusão seletiva dos votos aos membros de S está na origem da eficiência do protocolo.

6 Custo do Protocolo NB-2PC de Validação Atômica Fraca

Iremos analisar o custo do protocolo NB-2PC considerando um cenário favorável: não existem nem falhas, nem suspeitas equivocadas e todos os processos são favoráveis a validar a transação (votam *sim*). Na prática, este é o caso mais freqüente. Além disso, como explicamos anteriormente (Seção 3), em presença de falhas o nosso protocolo goza das mesmas vantagens associadas à utilização do *consensus* como protocolo de terminação. Nossa análise leva em conta o *número de passos de comunicação* e o *número de mensagens difundidas* para chegar ao resultado. A troca de mensagens será analisada em dois ambientes, diferenciados pelo tipo de primitiva de comunicação que suportam: i) comunicação ponto-a-ponto e ii) comunicação por difusão (*broadcast*).

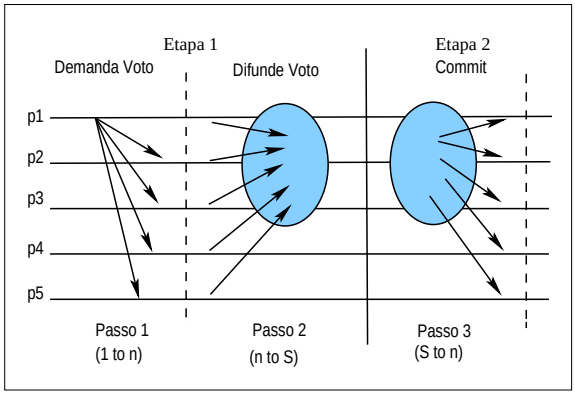

Figura 6: Passos do Protocolo NB-WAC, com *líder* $= p_1$ e $S = \{p_1, p_2, p_3\}$

A Figura 6 ilustra as fases e o passos de comunicação realizados pelo protocolo NB-2PC. Ele decide em *três passos de comunicação*. Num primeiro passo a transação é iniciada (execução do módulo 4). Em seguida, a função Atomic_Commit() é chamada e conclui em dois passos de comunicação: um passo para a divulgação dos votos (*sim* ou *não*) e outro passo para a propagação das proposições (COMMIT ou ABORT). O cálculo termina na ocorrência da seguinte condição: se o universo de processos em S adota a mesma proposição (todos votam seja por COMMIT, seja por ABORT). Esta exigência é facilmente satisfeita se o cenário favorável anteriormente descrito se produz.

Em relação ao número de mensagens trocadas, observemos que a cardinalidade do conjunto S deve ser superior a f. Logo, podemos considerar que $|S| = f + 1$. No total, o número de mensagens ponto a ponto propagadas para chegar ao COMMIT é igual a $n(2f + 3)$, o que equivale a $n + f + 2$ mensagens de difusão. Esse total representa adição dos seguintes valores:

- Primeiro passo: n mensagens ponto a ponto (ou 1 mensagem de difusão) – demanda de voto do líder a todos os demais;

- Segundo passo: $n(f+1)$ mensagens ponto a ponto (ou n mensagens de difusão) – difusão de votos de todos os processos aos membros de S;

- Terceiro passo: $n(f+1)$ mensagens ponto a ponto (ou $f+1$ mensagens de difusão) – primeira fase do *consensus*, difusão de proposições pelos processos de S a todos os outros.

  No cenário favorável considerado, todos os processos decidem nesse passo e não precisam esperar por mensagens de decisão vindas indiretamente de outros processos; sendo assim, o custo da utilização da primitiva de difusão confiável não é considerado no nosso cálculo (linhas 5 e 9 da figura 2).

6.1 Comparação com Trabalhos Similares

A Tabela 1 enumera os resultados de eficiência apresentados por alguns dos principais protocolos. Distingui-se cinco dentre eles: 2PC [8], 3PC [9, 10], [GS95] [11], [GLS96][12] e a nossa proposta [NB-2PC]. Nossa avaliação considera as condições favoráveis e as mesmas medidas de desempenho anteriormente descritas.

Protocolo	**# Passos Comunicação**	**# Mensagens Ponto a Ponto**	**# Mensagens Broadcast**
2PC	3	$3n$	$n+2$
3PC	5	$5n$	$2n+3$
[GS95]	3	$n(2n+1)$	$2n+1$
[GLS96]	3	$4nf+3n$	$n+2f+2$
[NB-2PC]	3	$2nf+3n$	$n+f+2$

Tabela 1: Comparação entre Diversos Protocolos de Validação Atômica

Como indicamos anteriormente, o 2PC é a solução mais eficiente: ela requer somente três passos de comunicação e a emissão de $3n$ mensagens ponto a ponto para decidir. Entretanto, ela não resolve o NB-WAC. Para resolver esse problema, os protocolos de 3PC realizam dois passos de comunicação a mais a fim de evitar as blocagens e difundem igualmente mais mensagens (no total 5 passos e $5n$ mensagens são necessárias).

Os trabalhos [11, 13, 14, 12] foram propostos a partir da utilização do serviço de *consensus* como um protocolo de terminação para tratar os problemas ligados a existência de falhas. Para esses algoritmos, não estamos considerando nos valores apresentados o custo da primitiva de difusão confiável utilizada para divulgar a decisão final. Este é, de fato, o procedimento corrente na literatura. Nesse protocolos, tais mensagens não são necessárias para decidir num cenário sem falhas; além disso, na prática, a complexidade do uso de tal primitiva pode ser minimizada [15].

[13] e [14] não admitem decisões antecipadas e terminam em ao menos *quatro* passos de comunicação. [GS95] [11] apresenta melhores resultados de desempenho. Nas condições favoráveis, ele termina em três passos e demanda a emissão de $n(2n+1)$ mensagens ponto a ponto. Ao nosso conhecimento, o protocolo proposto por [GLS96] [12] é até então, o mais eficiente dentre os protocolos baseados no *consensus*. Ele termina o cálculo em três passos de

comunicação e necessita a emissão de $4nf + 3n$ mensagens ponto a ponto (ou $n + 2f + 2$ mensagens de difusão). Entretanto, em comparação com o protocolo [NB-2PC], ele difunde $2nf$ mensagens ponto a ponto a mais[2].

O princípio adotado por [GLS96] se apoia igualmente sobre a idéia de terminação guiada pelos participantes. Um conjunto de processos, designado Set_{NB} e escolhido de antemão é o responsável pela terminação antecipada do cálculo. O papel atribuído a Set_{NB} é o mesmo que o do conjunto S utilizado na nossa solução. Os protocolos se distinguem pela forma como o conjunto é acionado e pelas condições sob as quais o cálculo termina. Nas duas soluções as mensagens da primeira fase (troca de votos *sim* e *não*) são difundidas somente aos processos pertencentes ao conjunto (S ou Set_{NB}). Em seguida, somente esses processos difundem as proposições de validação para a transação (COMMIT ou ABORT). Entretanto, se uma terminação antecipada não é possível nesta fase, em [NB-2PC] *todos* os processos iniciam numa terceira fase o *consensus*, ao tempo que em [GLS96] somente os processos de Set_{NB} participam do *consensus*. Essa diferença está na origem dos melhores resultados de desempenho obtidos por nossa solução, pois, para assegurar o término do *consensus* é preciso que $|Set_{NB}| \geq 2f + 1$, e no nosso protocolo $S \geq f + 1$.

Uma outra diferença entre os dois trabalhos é que [GLS96] só decide prematuramente se uma parte dos processos ($f + 1$) em Set_{NB} votam por COMMIT. Nossa solução permite uma decisão antecipada num caso mais genérico: desde que todos os processos ($f + 1$) de S votem igual (ou COMMIT ou ABORT). Essa característica é interessante quando existem suspeitas equivocadas ou falhas reais após a primeira fase de troca de votos. Isto é, todos os processos votam *sim* e em seguida, devido a ocorrência de falhas ou de suspeita de falhas, eles propõem ABORT ao *consensus*. Mesmo em tais condições nossa solução antecipa a decisão.

De um modo geral, os protocolos baseados no *consensus* exibem uma complexidade de mensagens ponto a ponto de $kn^2 + f(n)$ e de mensagens de difusão de $k'n$. Os melhores protocolos [GLS96] e [NB-2PC] exibem resultados que dependem do número de faltas f toleradas pelo sistema. No caso geral, onde $f < n/2$, é interessante de constatar que [NB-2PC] requer $n^2 + f(n)$ e $3/2n$, enquanto que todos os outros protocolos exibem respectivamente $2n^2 + f(n)$ e $2n$ como fator de complexidade para a emissão de mensagens ponto a ponto e de difusão. Por outro lado, em presença de um número pequeno de faltas ($f \leq 2$), [NB-2PC] é tão eficiente quanto o clássico 2PC numa rede a *broadcast* ou o 3PC numa rede ponto a ponto.

7 Conclusão

O problema da *validação atômica* é indispensável à implementação de transações em sistemas distribuídos, pois ele se ocupa de garantir a consistência dos dados na ocorrência de falhas. Neste trabalho, apresentamos um protocolo original de validação atômica não-bloqueante, baseado num algoritmo de *consensus*. A solução obtida é mais eficiente que qualquer outro protocolo equivalente proposto até então. Em presença de um baixo número de faltas ($f \leq 2$), ela é tão eficiente quanto o clássico 2PC (em quantidade de mensagens de difusão) ou o 3PC (em número de mensagens ponto a ponto). No geral ($f < n/2$), a sua complexidade absoluta é melhor que a complexidade dos melhores protocolos existentes. A estratégia de concepção adotada

[2] A definição de complexidade adotada por [GLS96] leva em conta o número de mensagens *recebidas* antes do COMMIT e não o número de mensagens *enviadas* para chegar ao mesmo. Dessa maneira, a complexidade anunciada é de $3nf + 3n$ mensagens ponto a ponto recebidas para chegar a decisão, no lugar de $4nf + 3n$. Nosso trabalho não leva em conta tal medida de complexidade por considerar que ela não reflete a quantidade real de mensagens que transitam na rede.

para a obtenção da solução é bastante modular. O *consensus* é um bloco independente, que deve ser regulado por parâmetros específicos, a fim de garantir uma terminação antecipada em condições favoráveis. O *framework* obtido para a resolução da validação atômica é suficientemente genérico para ser utilizado na solução de outros problemas de acordo. Tal possibilidade está sendo atualmente investigada pelos autores.

8 Agradecimentos

Os autores agradecem a contribuição de Michel Hurfin (IRISA) a este trabalho.

Referências

[1] F. Brasileiro, F. Greve, A. Mostefaoui, M. Raynal, Consensus in One Communication Step is Possible. In *PaCT (International Conference on Parallel Computing Technologies)*, Springer-Verlag LNCS 1800, pp. 1258-1265, September 2001.

[2] T. Chandra and S. Toueg, Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, March 1996.

[3] T. Chandra, V. Hadzilacos and S. Toueg, The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[4] M. Fischer, N. Lynch and M. Paterson, Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, April 1985.

[5] R. Guerraoui Revisiting the Relationship Between Non-Blocking Atomic Commitment and Consensus. In *9th Int. Workshop on Distributed Algorithms*, Springer-Verlag, LNCS 972, pp. 87–100, (J-M. Hélary and M. Raynal Eds), Le Mont-Saint-Michel (France), September 1995.

[6] F. Greve, M. Hurfin, M. Raynal, F. Tronel, Primary Component Asynchronous Group Membership as an Instance of a Generic Agreement Framework. *ISADS'2001: 5th International Symposium on Autonomous Decentralized Systems*, pp 93-100, March 2001.

[7] R. Guerraoui, A. Schiper The Generic Consensus Service. *IEEE Transactions on Software Engineering*, 27(1), pp. 29-41, January 2001.

[8] J. Gray, Notes on Database Operating Systems. In *Operating Systems An Advanced Course*, pp 10-17. Lecture Notes in Computer Science (60), Springer-Verlag. 1978.

[9] D. Skeen, NonBlocking Commit Protocols. In *ACM SIGMOD International Conference on Management of Data*, pp 133-142. ACM Press. 1981.

[10] I. Keider, D. Dolev Increasing the Resilience of Atomic Commit, at No Additional Cost. In *ACM PODS'1995: Principles of Database Systems*, pp 245-254, May 1995.

[11] R. Guerraoui, A. Schiper, The Decentralized Non-Blocking Atomic Commitment Protocol. In *14th IEEE International Symposium on Parallel and Distributed Processing*, pp. 2-9, San Antonio, October 1995.

[12] R. Guerraoui, M. Larrea, A. Schiper, Reducing the Cost for Non-Blocking in Atomic Commitment. In *16th IEEE International Conference on Distributed Computing Systems (ICDCS-16)*, pp. 692-697, Hong-Kong, May 1996.

[13] M. Raynal, Revisiting the Non-Blocking Atomic Commitment Problem in Distributed Systems. In *2nd Workshop on Fault-Tolerant Parallel and Distributed Systems*, 1997.

[14] M. Hurfin, F. Tronel, A Solution to Atomic Commitment Based on an Extended Consensus Protocol. In *6th IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 98-103, 1997.

[15] L. Rodrigues, P. Veríssimo, Topology-Aware Algorithms for Large Scale Communication. In *Advances in Distributed Systems*, Springer-Verlag, LNCS 1752, pp. 1217-1256, 2000.

A Prova de Correção do Protocolo

Um aspecto interessante do uso do *consensus* como serviço de base para a construção de soluções para outros problemas de acordo é que a corretude das propriedades associadas a tais problemas se apoia nas propriedades do *consensus* sub-jacente. Desta maneira, as provas apresentadas para os problemas gozam das mesmas vantagens ligadas à sua concepção: modularidade e fácil compreensão. Na demonstração que se segue, estamos assumindo a correção do protocolo Set_Consensus [1].

Teorema 1 AC_Validade*: se um processo decide* COMMIT*, então todos os processos votaram* sim.

Prova Nas tarefa $T2$ (linha 9) e $T1$ (linha 2), o módulo Atomic_Commit só decide ABORT. As demais decisões são consequência da execução do Set_Consensus nas linhas 6, 7 e 8. Somente na linha 6 o valor COMMIT é proposto e somente se a seguinte condição se verifica: todos os processos de Π possuem votam *sim* (linha 5). A propriedade de C_Validade do serviço de *consensus* garante que um valor decidido é um dos valores propostos. Isto comprova o Teorema. $\square_{Teorema\ 1}$

Teorema 2 AC_Integridade*: um processo decide no máximo uma vez;*

Prova Esta propriedade é uma consequência direta da respectiva propriedade C_Integridade do *consensus* e da forma como a decisão é tratada pela função: retorno imediato do valor decidido (linhas 2 e 9). $\square_{Teorema\ 2}$

Teorema 3 AC_Acordo_Uniforme*: dois processos não decidem diferentemente;*

Prova

- No módulo Atomic_Commit, um processo só decide ABORT na tarefa $T2$ (linha 9), se um outro processo votou *não* e consequentemente decidiu por ABORT na tarefa $T1$ (linha 2).
- Se algum processo decidiu ABORT na linha 2, pelo Teorema 1, aqueles processos que continuam a execução da função e invocam o Set_Consensus, propõem seja o ABORT (se eles pertencem a S) (linha 7) seja o $\perp$ (aqueles que não estão em S). Como somente as proposições dos processos em S são consideradas na execução de Set_Consensus (linhas 1, 2 e 6), os processos que decidem no módulo de *consensus*, decidirão por ABORT.

- Se nenhum processo decidiu na linha 2, todos os processos corretos continuam a execução da função e invocam o Set_Consensus. Pela propriedade de C_Acordo_Uniforme deste módulo, dois processos não decidem diferentemente e o Teorema segue.

$\square_{Teorema\ 3}$

Teorema 4 AC_Terminação*: todos os processos corretos decidem de maneira definitiva;*

Prova As únicas instruções que podem impedir um processo correto de continuar a executar o protocolo correspondem as:

- Linhas 3 (módulo de Transaction) e 4 (módulo de Atomic_Commit). Pela propriedade de completude forte do detector de falhas $\diamond\mathcal{S}$, todo processo faltoso finirá por ser suspeito. Além disso, todo processo inicia a função Atomic_Commit pela emissão do seu voto (linha 1), como os canais são confiáveis, todo voto de um processo correto finirá por ser recebido. Logo, os processos corretos não ficam bloqueados na execução dessas instruções.

- Chamadas do Set_Consensus (linhas 6-8). Pela propriedade de C_Terminação deste módulo, todos os processos corretos decidem de maneira definitiva. Sendo assim, o Teorema segue.

$\square_{Teorema\ 4}$

Teorema 5 AC_Não_Trivialidade*: se todos os participantes votam* sim *e não existem* suspeitas de falhas *então a decisão é* COMMIT.

Prova Para realizar essa prova, iremos mostrar que o valor ABORT só pode ser decidido se algum processo vota *não* ou se suspeita da falha de outro processo. Na função Atomic_Commit, um participante só pode decidir ABORT nas linhas 2, 9 e na execução da função Set_Consensus (linhas 6-8).

- Um processo só decide na linha 2 se ele votou *não*;

- Se ele decide na linha 9, algum outro processo executou a linha 2 e portanto votou *não*;

- Na função Set_Consensus, para que o valor ABORT seja decidido, pela propriedade de C_Validade, é necessário que algum dos processos tenha votado ABORT; mas isso só ocorre se algum processo executou a linha 7. Nesse caso, ele suspeitou de algum outro processo (linhas 4-5). Isto comprova o Teorema.

$\square_{Teorema\ 5}$