Analisando o Custo do Armazenamento Tolerante a Faltas Bizantinas: PAXOS × Sistemas de Quóruns*

Alysson Neves Bessani, Wagner Saback Dantas,
Eduardo Adílio Pelison Alchieri, Joni da Silva Fraga

[1]DAS - Departamento de Automação e Sistemas
UFSC - Universidade Federal de Santa Catarina
Florianópolis - Santa Catarina

{neves,wagners,alchieri,fraga}@das.ufsc.br

Resumo. *A manutenção da disponibilidade e da integridade das informações é um requisito fundamental em sistemas de armazenamento de dados tolerantes a faltas. Uma abordagem comumente utilizada para concretização destes sistemas é a replicação tolerante a faltas Bizantinas. O presente trabalho avalia duas técnicas que podem ser utilizadas para a implementação de sistemas de armazenamento tolerantes a faltas Bizantinas: replicação máquina de estados, baseada no algoritmo* PAXOS *Bizantino, e sistemas de quóruns Bizantinos. Nossa abordagem compara teórica e experimentalmente duas implementações de um serviço de armazenamento, cada uma baseada em uma destas técnicas. Os resultados demonstram claramente as vantagens e desvantagens destas abordagens quando consideramos uma rede local e um número mínimo de réplicas.*

1. Introdução

A manutenção da disponibilidade e da integridade das informações é um requisito fundamental em sistemas de armazenamento de dados. Muitos destes sistemas devem manter estas propriedades mesmo em face à ocorrência de faltas acidentais ou intencionais (maliciosas), sendo que estas últimas são particularmente preocupantes uma vez que se originam de ataques bem sucedidos que levam a intrusões no sistema de armazenamento. A fim de prover armazenamento que tolere faltas acidentais e maliciosas, podemos considerar que o sistema está sujeito a **faltas Bizantinas** [Lamport et al., 1982] (a classe mais abrangente de faltas) e então empregar técnicas de tolerância a faltas Bizantinas (TFB) em sua concretização.

Duas técnicas podem ser utilizadas para implementar replicação visando TFB em sistemas de armazenamento: a **Replicação Máquina de Estados** (RME) [Schneider, 1990, Castro and Liskov, 2002], uma técnica mais geral que permite a construção de implementações tolerantes a faltas Bizantinas para qualquer serviço determinista. Esta técnica se fundamenta na utilização de protocolos de acordo para garantir que todas as réplicas do serviço executem a mesma sequência de operações; os **Sistemas de Quóruns** [Malkhi and Reiter, 1998] são uma técnica específica para implementação de armazenamento que se fundamenta na execução de leituras e escritas de dados em diferentes conjuntos de servidores que se intersectam.

As diferenças entre estas duas técnicas podem ser resumidas em dois pontos: (a) replicação máquina de estados pode ser utilizada na implementação de qualquer serviço determinista, enquanto sistemas de quóruns podem implementar apenas armazenamento

*Projeto apoiado pelo CNPq (processo 550114/2005-0).

(operações de leitura e escrita); (b) replicação máquina de estados requer a resolução de consenso, o que exige algumas premissas do ambiente (ou protocolos com terminação probabilista) [Fischer et al., 1985], enquanto sistemas de quóruns podem ser implementados em sistemas assíncronos. Estas diferenças têm fomentado um debate na comunidade de sistemas distribuídos a respeito da "ineficiência" do modelo de máquina de estados e da busca por alternativas a este modelo, dentre as quais se destacam os sistemas de quóruns Bizantinos [Abd-El-Malek et al., 2005, Ekwall and Schiper, 2005]. Trabalhos recentes têm explicitado as vantagens e desvantagens destas técnicas quando comparadas, exaltando tanto o caráter geral da máquina de estados [Ekwall and Schiper, 2005] quanto a possibilidade de implementação dos sistemas de quóruns com quase nenhuma premissa [Zhou et al., 2002] e sua potencial escalabilidade [Abd-El-Malek et al., 2005].

A literatura sobre a construção de sistemas tolerantes a faltas Bizantinas tem apresentado alguns avanços interessantes no que tange a ambas as técnicas, dentre os quais podemos citar: a demonstração de que a replicação máquina de estados tolerante a faltas Bizantinas pode ser implementada de forma eficiente [Castro and Liskov, 2002, Moniz et al., 2006]; as novas otimizações descobertas para o protocolo de consenso PAXOS Bizantino [Martin and Alvisi, 2005, Zielinski, 2004]; e os novos protocolos para sistemas de quóruns Bizantinos que toleram clientes maliciosos utilizando um número ótimo de servidores [Liskov and Rodrigues, 2006, Cachin and Tessaro, 2006]. Estes avanços sugerem que as duas técnicas podem ser utilizadas na implementação de serviços confiáveis. Eles também instigam algumas perguntas: qual destas técnicas é a mais eficiente? Em que condições uma destas técnicas deve ser usada em detrimento a outra?

Neste trabalho, investigamos esta questão através da avaliação experimental de dois dos protocolos mais eficientes e completos para concretização destas técnicas: PAXOS Bizantino [Castro and Liskov, 2002] (replicação máquina de estados) e BFT-BC (*Byzantine Fault Tolerance with Byzantine Clients*) [Liskov and Rodrigues, 2006] (sistemas de quóruns). A abordagem adotada é pragmática: implementamos um serviço de armazenamento replicado usando ambas as técnicas e comparamos as características de ambos os protótipos experimentalmente considerando *(i.)* um sistema com 4 servidores onde, no máximo, um deles pode falhar de forma arbitrária, *(ii.)* a implementação de um serviço de armazenamento de dados, onde blocos de dados[1] podem ser lidos e escritos e *(iii.)* assumimos uma rede local com elevada capacidade de comunicação. A premissa *(i.)* se deve ao alto custo de se implementar independência de falhas para um grande número de réplicas [Obelheiro et al., 2005]. A premissa *(ii.)* é motivada pelo fato de o serviço de armazenamento implementar a abstração de um registrador atômico [Lamport, 1986], o tipo mais forte de objeto que um sistema de quóruns, que não requer consenso, pode implementar [Herlihy, 1991]. A premissa *(iii.)* se deve principalmente a complexidade inerente de se implementar e gerenciar este tipo de serviço em uma rede de larga escala.

Considerando este ambiente, vários testes são executados em diversas condições de carga (processos executando operações concorrentemente) e falha nos servidores. Estes testes demonstram os pontos fortes e fracos dos protocolos examinados quando executados em um ambiente como o definido pelas premissas já descritas. Estes testes ainda dão importantes pistas de qual técnica é mais adequada para que tipo de ambiente.

[1] Os protótipos implementados suportam armazenamento de objetos Java serializados.

2. Modelo de Sistema

Assumimos um sistema com 4 servidores e um número ilimitado de clientes. Deste conjunto de processos, um dos servidores (respeitando o limite teórico de que o número de servidores n deve ser pelo menos $3f + 1$ para que o sistema tolere f faltas Bizantinas [Martin et al., 2002, Toueg, 1984]) e um número qualquer de clientes pode falhar de forma arbitrária. Todos os processos se comunicam através de canais ponto a ponto bidirecionais confiáveis e autenticados. Desta forma, o modelo de faltas adotado é o de **faltas Bizantinas com autenticação** [Lamport et al., 1982].

Em termos de premissas temporais, assumimos um sistema completamente assíncrono tanto em termos de computação quanto em termos de comunicação. Devido à conhecida impossibilidade FLP [Fischer et al., 1985], o algoritmo de replicação máquina de estados utilizado não garante terminação nesse modelo. No entanto, como nossos experimentos são executados apenas em rede local, onde é sabido que esta impossibilidade não tem efeito prático [Urbán et al., 2001], o algoritmo empregado sempre termina.

O serviço de armazenamento desenvolvido é modelado através de um **registrador atômico** [Lamport, 1986] r, suportando as operações $r.write(v)$, que escreve um valor v em r, e $r.read()$, que devolve o valor atual do registrador r. Estas operações são invocadas remotamente pelos processos clientes nos registradores implementados nos servidores. Os servidores corretos não permitem a atualização dos registradores de outra forma que não seja seguindo os protocolos.

3. As Técnicas Comparadas

Esta seção detalha as duas técnicas comparadas e os protocolos empregados na implementação do serviço de armazenamento com cada uma destas técnicas.

3.1. Replicação Máquina de Estados

A replicação máquina de estados [Schneider, 1990] é o método mais empregado na concretização de sistemas distribuídos tolerantes a faltas Bizantinas. A implementação deste tipo de replicação requer **determinismo de réplicas**: partindo de um mesmo estado inicial, os estados de todas as réplicas devem ter a mesma evolução, i.e., após executar a mesma sequência de operações, as réplicas devem alcançar o mesmo estado [Schneider, 1990]. Este requisito exige que o sistema tenha as seguintes propriedades: *(i.)* difusão confiável com ordem total das requisições; *(ii.)* os estados das réplicas são alterados apenas pela execução de requisições; *(iii.)* as operações executadas nas réplicas devem ser deterministas. As propriedades *(ii.)* e *(iii.)* são garantidas diretamente pelas propriedades do sistema: um registrador é, por definição, determinista, e os servidores implementados ignoram qualquer pedido de alteração do valor do registrador que não seja enviado seguindo o protocolo de escrita. Desta forma, resta-nos definir um protocolo para difusão com ordem total para satisfazer a propriedade *(i.)*. O ponto central para concepção deste protocolo reside na resolução do **problema de consenso**.

Em um sistema distribuído composto por diversos processos independentes, o **problema do consenso** consiste em fazer com que todos os processos corretos acabem por decidir (Terminação) o mesmo valor (Acordo), o qual deve ter sido previamente proposto por algum dos processos do sistema (Validade[2]). O protocolo de consenso usado em nossa implementação de replicação máquina de estados é o **PAXOS Bizantino** [Castro and Liskov, 2002] com modificações para terminação rápida (em dois passos

[2]Esta é apenas uma das possíveis definições da propriedade de Validade.

de comunicação) [Martin and Alvisi, 2005, Zielinski, 2004], doravante chamado apenas de PAXOS. Este protocolo foi escolhido devido ao seu bom desempenho em casos livres de falha e à sua resiliência ótima.

O algoritmo PAXOS considera três classes de agentes: **proponentes**, os quais propõem os valores; **aceitantes**, os quais escolhem um único valor entre os propostos; **aprendizes**, os quais precisam aprender o valor decidido. Em nossa implementação, todos os servidores do sistema desempenham estes três papéis ao mesmo tempo; no entanto, distinções serão feitas para facilitar o entendimento do protocolo.

Este algoritmo é executado em *rounds*, sendo que, em cada *round* r, um proponente p_r é escolhido líder. Este líder tem a responsabilidade de escolher e enviar uma proposta aos aceitantes, os quais tentarão fazer deste valor a decisão do consenso através de uma ou mais fases de trocas de mensagens visando garantir o Acordo. Por fim, quando estabelecida, a decisão de consenso é enviada aos aprendizes. As propriedades de segurança sempre são mantidas pelo protocolo, mas o consenso somente terá progresso em *rounds* favoráveis. Um *round* é considerado **favorável** quando seu líder é correto (cada round tem apenas um líder, o processo $r\%4$) e o sistema está num periodo de sincronia: as comunicações e computações ocorrem dentro de um período de tempo limitado. Nesta situação, um valor proposto pode ser aprendido dentro do período de um *round*. Adicionalmente, um *round* é dito **muito favorável** se ele é favorável e não existem falhas nos aceitantes. Caso um *round* r não seja favorável, um novo *round* é iniciado com um novo líder e assim sucessivamente até que um valor seja aprendido.

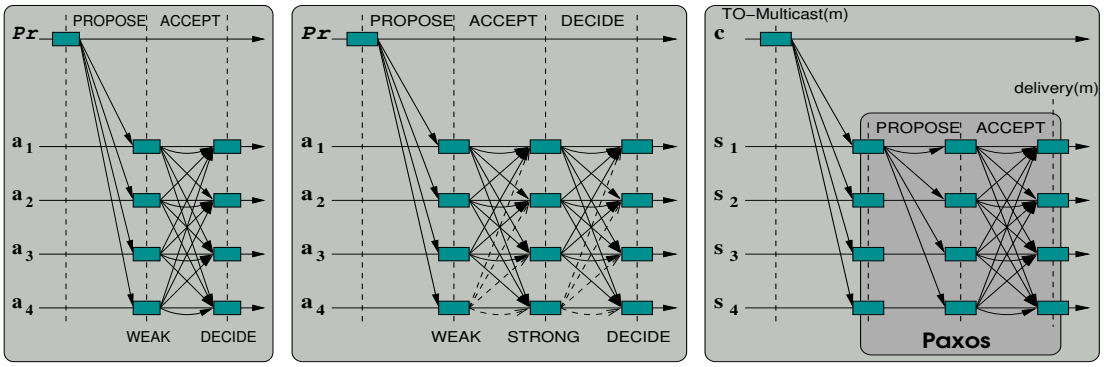
As Figuras 1(a) e 1(b) ilustram alguns cenários de execução do PAXOS Bizantino A Figura 1(a) mostra uma execução, em que o protocolo executa um *round* muito favorável e consegue terminar em apenas dois passos[3]. Este padrão segue as otimizações definidas em [Martin and Alvisi, 2005, Zielinski, 2004]. O caso normal de operação do PAXOS, onde o primeiro *round* é favorável, é apresentado na Figura 1(b): um *round* do algoritmo consolida uma decisão em três passos de comunicação [Castro and Liskov, 2002]. Caso um *round* não seja completado em um determinado intervalo de tempo, um novo *round* é iniciado através de um protocolo de transição, que requer dois passos de comunicação. Este protocolo de troca é o único passo do protocolo onde criptografia de chave pública é necessária; logo, o protocolo não requer este mecanismo em execuções favoráveis.

A implementação da difusão com ordem total usando o PAXOS se baseia na execução de uma instância deste algoritmo para cada mensagem a ser ordenada. Desta forma, uma requisição m é a i-ésima requisição a ser executada se e somente se for o resultado da execução i do PAXOS [Castro and Liskov, 2002, Martin and Alvisi, 2005]. A Figura 1(c) ilustra a difusão com ordem total de uma requisição usando o PAXOS em uma execução muito favorável[4].

Uma otimização usualmente implementada em replicação máquina de estados é a tentativa de execução de algumas operações sem a necessidade de execução do protocolo de ordem total. Com esta otimização, toda operação que não altere o estado do serviço (uma leitura, por exemplo) é enviada aos servidores, que respondem imediatamente. Se o cliente obtém $n-f$ respostas iguais, a operação termina; caso contrário, a requisição é reenviada através da difusão com ordem total. Esta otimização permite que uma leitura seja

[3]Note que, como os próprios aceitantes são os aprendizes, não é necessário difundir o valor decidido para estes últimos.

[4]Nesta figura não consideramos o envio das respostas pelos servidores para o cliente.



(a) Favorável. (b) Favorável com a_4 faltoso. (c) Difusão com ordem total.

Figura 1. Execuções do PAXOS.

completada em dois passos de comunicação (envio e resposta) em ocasiões onde não existem faltas ou operações de escrita sendo executadas concorrentemente. A implementação da operação de leitura de nosso serviço de armazenamento usa essa otimização.

Como consideramos faltas Bizantinas, um líder (proponente) malicioso pode propor uma operação inexistente para execução, ferindo a vivacidade do sistema e fazendo com que os servidores fiquem bloqueados esperando o recebimento desta requisição. A resolução deste problema é descrita em [Castro and Liskov, 2002]: o líder deve ser trocado, e requisições inexistentes já ordenadas devem ser definidas como operações *nop*, que não alteram o estado do sistema.

3.2. Sistemas de Quóruns Bizantinos

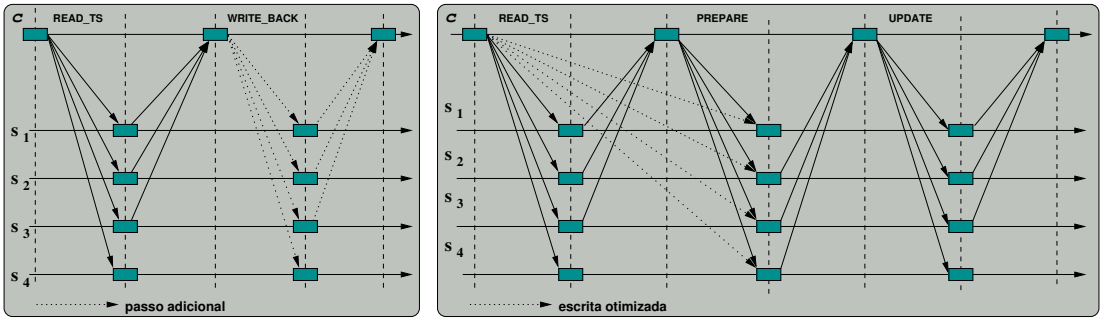
Sistemas de quóruns Bizantinos [Malkhi and Reiter, 1998], doravante denominados apenas como sistemas de quóruns, implementam sistemas replicados de armazenamento de dados distribuídos com garantias de consistência e disponibilidade mesmo com a ocorrência de faltas Bizantinas em algumas de suas réplicas. Ao contrário dos sistemas baseados em replicação máquinas de estados, um protocolo para sistemas de quóruns não exige a execução de acordo entre as réplicas do serviço, o que livra esta solução da impossibilidade FLP [Fischer et al., 1985] e permite sua implementação com terminação garantida em sistemas assíncronos. Algoritmos para sistemas de quóruns são reconhecidos por seus bons desempenho e escalabilidade, já que os clientes desse sistema acessam de fato somente um conjunto particular de servidores ao invés de todos os servidores.

Servidores em um sistema de quóruns organizam-se em subconjuntos denominados **quóruns**. Cada dois quóruns de um sistema mantém um número suficiente de servidores corretos em comum (garantia de consistência), sendo que existe pelo menos um quórum no sistema formado somente por servidores corretos (garantia de disponibilidade) [Malkhi and Reiter, 1998]. Os clientes realizam suas operações em registradores de leitura e escrita replicados por estes quóruns, cujos tamanhos para operações de leitura e escrita podem ser iguais (**quóruns simétricos**) ou não (**quóruns assimétricos**). Cada registrador detém um par $\langle v, t \rangle$ com um valor v do dado armazenado e uma estampilha de tempo (*timestamp*) t associada. Este *timestamp* é definido pelo cliente quando da sua escrita, sendo que cada cliente c utiliza conjuntos disjuntos de *timestamps*.

Dentre as muitas construções e protocolos para sistemas de quóruns que têm sido propostas nos últimos anos (por exemplo, [Malkhi and Reiter, 1998, Martin et al., 2002, Liskov and Rodrigues, 2006]), escolhemos para este trabalho o BFT-BC, um algo-

ritmo bastante recente que apresenta resiliência ótima e tolera clientes faltosos [Liskov and Rodrigues, 2006]. Esta solução requer, para até f faltas Bizantinas no sistema, uma quantidade de $n \geq 3f + 1$ servidores, com quóruns de $2f + 1$ servidores e intersecções com $f + 1$ servidores (pelo menos um correto). Cada servidor emula um registrador atômico de semântica compatível com um registrador implementado pela máquina de estados e é resistente a uma grande extensão de problemas possivelmente suscitados por clientes Bizantinos e não contemplados em trabalhos anteriores do gênero.

A fim de manter suas semânticas de consistência, o BFT-BC utiliza um mecanismo de provas assinadas em todas as suas etapas de execução. Desta maneira, para o cliente ingressar em uma nova fase do algoritmo, é preciso que ele apresente uma prova de que completou a fase anterior. Esta prova nada mais é que o conjunto das mensagens (assinadas) de resposta coletadas de um quórum de servidores na fase anterior. (por exemplo, para o cliente escrever no quórum, é preciso que ele tenha terminado um escrita anterior). Através do uso desta técnica, o BFT-BC emprega, na leitura, 2 ou 4 passos de comunicação (Figura 2(a)) e 4 ou 6 passos na escrita (Figura 2(b)).



(a) Leitura (b) Escrita

Figura 2. Execuções de operações usando o BFT-BC.

De maneira geral, o algoritmo BFT-BC executa da seguinte forma: na **leitura**, o cliente inicialmente requisita um conjunto de pares $\langle v, t \rangle$ válidos de um quórum e seleciona aquele com o maior *timestamp*. Caso todos os pares retornados possuam este mesmo maior *timestamp* e um mesmo valor (o que ocorre em execuções sem escritas concorrentes e sem falhas), a execução termina. Caso contrário, o cliente realiza um passo adicional de reescrita (*write back*) no sistema e espera confirmações de um quórum; para **escrita**, o cliente tem duas opções. Em um cenário normal, o cliente obtém um conjunto de *timestamps* de um quórum como na leitura, depois realiza a preparação de sua escrita, fase em que tenta obter de um quórum um conjunto de provas necessário para completar sua operação. Em caso de sucesso nesta preparação, o cliente escreve no quórum, esperando um conjunto de confirmações. Em um cenário alternativo, o cliente pode, em vez da escrita normal, executar um protocolo otimizado de escrita, realizando em uma única fase as etapas de coleta de *timestamp* e de preparação da escrita, como mostrado pelas linhas pontilhadas da Figura 2(b).

3.3. Comparando as Técnicas

A Tabela 1 compara as implementações do serviço de armazenamento. Os valores da tabela mostram que, se considerarmos apenas as métricas de comunicação, usualmente empregadas na avaliação de algoritmos distribuídos, o protocolo para sistemas de quóruns (BFT-BC) é igual ou superior à replicação máquina de estados baseada no PAXOS: mesma

quantidade de passos de comunicação no melhor caso em ambas as operações, quantidade de passos de comunicação limitada no pior caso (uma execução do PAXOS pode requerer vários *rounds*) e complexidade de mensagens linear (ao invés de quadrática).

Característica	Operação	PAXOS	BFT-BC
Modelo de Sistema	-	parcialmente síncrono	assíncrono
Número de Servidores	-	$3f + 1$	$3f + 1$
Passos de Comunicação	**write**	4/5/*	4/6
	read	2/6/7/*	2/4
Complexidade das Mensagens	**write**	$O(n^2)$	$O(n)$
	read	$O(n)$/$O(n^2)$	$O(n)$
Necessidade de Assinatura	**write**	troca de líder	sempre
	read	troca de líder	*write back*
Necessidade de Verificação	**write**	troca de líder	sempre
	read	troca de líder	sempre

Tabela 1. Comparação entre a replicação usando o PAXOS e o BFT-BC.

Entretanto, se analisarmos o uso de criptografia assimétrica, a tabela mostra que o PAXOS só usa esse mecanismo em caso de troca de líder (o que ocorre apenas em execuções em períodos de assincronia ou com falha no processo líder), enquanto o BFT-BC requer assinatura e verificação em quase todas as execuções de ambas as operações.

4. Implementação e Ambiente de Execução

Os algoritmos apresentados na seção anterior foram implementados utilizando o arcabouço para prototipação, simulação e execução de algoritmos distribuídos NEKO [Urbán et al., 2002].

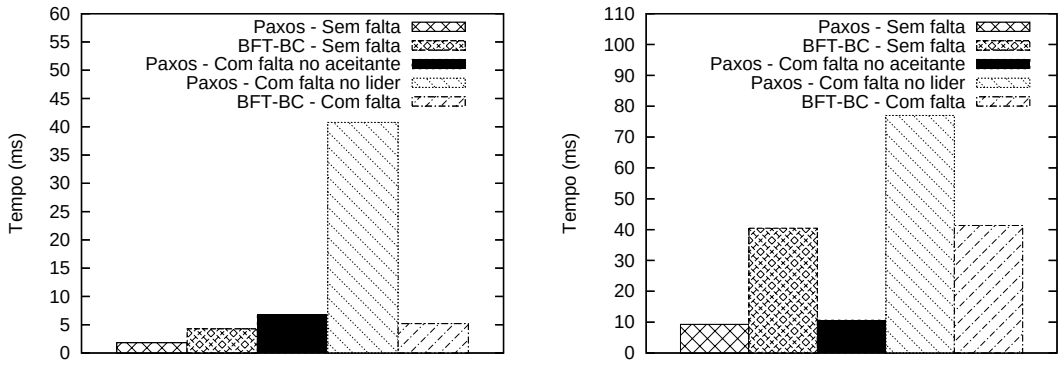
Os canais confiáveis e autenticados do sistema são implementados através do uso *sockets* TCP e chaves de sessão baseadas no algoritmo *HmacSHA-1*. As assinaturas utilizadas nos protocolos são implementadas usando os algoritmos *SHA-1* e *RSA* (1024 bits) para resumos e criptografia assimétrica, respectivamente. Para implementação desta criptografia foi utilizada a biblioteca padrão do Java 5.0 (*Java Cryptography Extensions*).

O ambiente de testes consta de 5 máquinas com a mesma configuração de *hardware* (AMD Athlon XP 1.9Ghz, 512MB de RAM, placa *ethernet* de 100MB/s) conectadas por um *switch* 1GB/s. O ambiente de *software* em todas as máquinas é também homogêneo: S.O. GNU/Linux *kernel* 2.6.12, máquina virtual Java da SUN versão 1.5.0_06.

5. Experimentos e Resultados

Esta seção apresenta os experimentos realizados e seus resultados. Todos os valores reportados aqui compreendem o tempo médio necessário (em milisegundos) para a execução de uma operação por um cliente do sistema, recolhido a partir de 1000 repetições. Os valores em tempo referidos no texto a seguir são aproximados.

A Figura 3 apresenta o latência média para execução de operações nos sistemas sob diversas condições de faltas e sem concorrência. Na Figura 3(a), são apresentados os resultados para a operação de leitura. Nos casos sem falha, esta operação é extremamente eficiente usando qualquer uma das duas técnicas: 2 ms para replicação máquina de estados e 4 ms para o sistema de quóruns. Isto se deve ao fato de ambas requererem apenas dois passos de comunicação. A diferença de 2 ms a favor da replicação máquina de estados se deve à não necessidade de verificação criptográfica dos valores lidos dos servidores



(a) Operação de leitura.

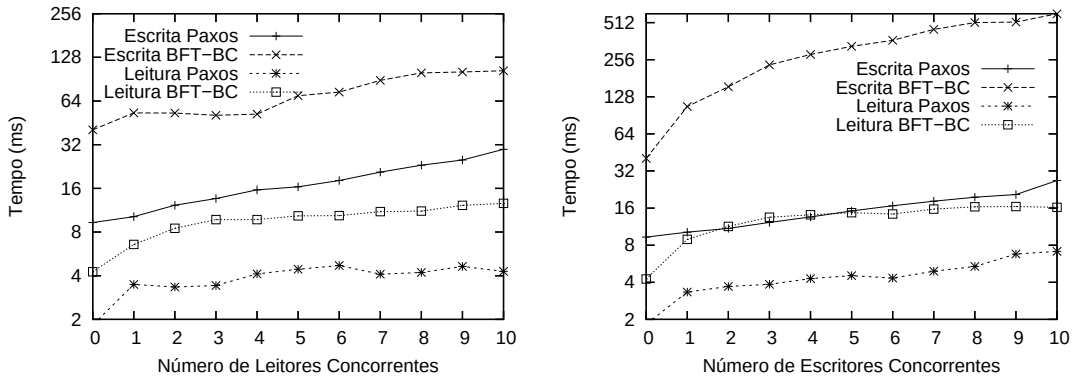
(b) Operação de escrita.

Figura 3. Desempenho dos protocolos de leitura e escrita.

consultados. A existência de um servidor faltoso não causa muito impacto no BFT-BC, pois o protocolo básico não se altera e o único processamento extra é a verificação de mais uma mensagem (uma vez que a mensagem inválida do servidor faltoso é descartada). Já no caso do PAXOS, o impacto é maior, pois algumas vezes o cliente pode coletar a resposta inválida do servidor malicioso e ser forçado a executar uma leitura com ordem total. Se o servidor malicioso for também o proponente do *round* inicial no PAXOS, o impacto é muito maior, já que a execução será levada para um segundo *round*. Em nossos experimentos, o servidor malicioso consegue evitar o término da escrita otimizada (sem necessidade de execução do PAXOS) em 50% dos casos. Note que a implementação de canais autenticados é bastante leve, uma vez que a geração e verificação de MACs para autenticação de mensagens leva menos de 1 ms.

Os resultados para a operação de escrita são apresentados na Figura 3(b). Nesta figura, é possível perceber que a escrita usando replicação máquina de estados é muito mais eficiente que o protocolo de escrita do BFT-BC na maioria dos casos. Isto se deve à acentuada latência imposta pela computação de uma assinatura *RSA* nos servidores na fase de preparação da escrita (esta operação demora em torno de 14 ms). Já no caso da replicação máquina de estados, este tipo de operação só é usado na troca de *rounds* (caso "Paxos - Com falha no líder"). Note que a diferença de tempo entre a execução rápida do PAXOS, em *rounds* muito favoráveis, e a execução normal (onde existem aceitantes faltosos) é praticamente inexistente. Isto se deve à diminuta latência para envio de mensagens via o canal autenticado em nosso modelo, o que torna o tempo requerido para a execução de um passo de comunicação extra praticamente irrisório. O único cenário em que a latência da replicação máquina de estados é (muito) maior que a do sistema de quóruns é quando o proponente do primeiro *round* do PAXOS é o servidor faltoso. Neste caso, a invocação de escrita só será ordenada no segundo *round*, o que acarreta uma latência total de quase 80 ms. Note ainda que a existência de um servidor faltoso não altera em praticamente nada a latência do protocolo de escrita do BFT-BC pelos mesmos motivos expostos na análise da operação de leitura.

A Figura 4 mostra a latência das operações em execuções sem faltas e com concorrência de 0-10 clientes executando operações de leitura (Figura 4(a)) e escrita (Figura 4(b)). O objetivo destes experimentos é verificar a escalabilidade dos sistemas apresentados quando expostos a uma alta carga de requisições.



(a) Desempenho com leitores concorrentes. (b) Desempenho com escritores concorrentes

Figura 4. Desempenho dos protocolos considerando operações concorrentes.

Conforme pode ser observado nas figuras, as operações executadas no serviço implementado usando replicação máquina de estados têm uma latência média sempre melhor que suas contrapartes implementadas utilizando sistemas de quóruns.

No caso das operações de leitura, ambas as técnicas apresentam boa escalabilidade, não importando o tipo de operação executada concorremente. A replicação máquina de estados apresentou latências entre 2 e 5 ms com leituras concorrentes e 2 a 8 com escritas concorrentes. O sistema de quóruns apresentou latências a partir de 4 ms e chegando a 14 ms com 10 leitores concorrentes e 16 ms com o mesmo número de escritores operando concorrentemente à leitura. Os aumentos dos valores nestes protocolos ocorrem justamente devido à contenção nos processadores dos servidores. A explicação para o aumento da diferença de latência entre a leitura com o BFT-BC e a mesma operação com a replicação máquina de estados pode ser derivada diretamente desta observação uma vez que o BFT-BC utiliza muito mais processador (usa mais criptografia) que o PAXOS. Um observação interessante é que o passo de reescrita do protocolo de escrita BFT-BC foi executado muito raramente nos experimentos, no máximo em $0,3\%$. Isto comprova que apesar desta fase requerer criptografia assimétrica, seu uso é raríssimo na prática.

Quando se trata da latência da operação de escrita executada com concorrência, dois pontos devem ser ressaltados. Em primeiro lugar, a replicação máquina de estado mantém uma boa escalabilidade mesmo quando o número de operações concorrentes cresce. As Figuras 4(a) e 4(b) atestam este fato já que a latência de processamento desta operação varia entre 8 e 32 ms não importando que operação é executada concorrentemente a escrita. Esta escalabilidade se deve ao fato de nossa implementação ordenar as requisições em lote (*batch*) periodicamente, não executando um consenso pra cada requisição, mas sim um consenso para o conjunto de requisições recebidas em um intervalo de tempo (se este conjunto não for vazio). O segundo ponto a ser ressaltado diz respeito à acentuação da contenção no processador, o que causa um aumento substancial da latência da escrita no protocolo BFT-BC. Quando a escrita ocorre concorrentemente a outras leituras, um crescimento linear na latência da operação de escrita é observado, variando de 40 ms (sem concorrência) a pouco mais de 100 ms (10 leitores concorrentes), conforme pode ser observado na Figura 4(a). O problema se agrava muito quando consideramos escritas concorrentes, uma vez que a quantidade de assinaturas produzidas nos servidores causa uma explosão na latência do protocolo. A Figura 4(b) mostra que de uma latência de 40

ms no cenário sem concorrência, o sistema chega a apresentar uma latência de 600 ms com 10 escritores concorrentes. Esta resultado é preocupante uma vez que no mesmo cenário a escrita na replicação máquina de estados apresenta latência de apenas 30 ms.

6. Trabalhos Relacionados

Tendo em vista a profusão de trabalhos sobre replicação máquina de estados e sistemas de quóruns, tanto considerando tolerância a faltas por parada quando Bizantinas, ainda é escassa a literatura que compara estas duas abordagens.

O primeiro trabalho a discutir especificamente o relacionamento e as diferenças entre estas duas abordagens é [Ekwall and Schiper, 2005]. Neste trabalho, os autores discutem, sob o prisma da serialização de operações[5], a relação entre sistemas de quóruns e replicação máquina de estados em sistemas sujeitos a faltas de parada. A conclusão fundamental deste trabalho é que os sistemas de quóruns somente suportam execuções atômicas de grupos de operações de leitura e escrita em sistemas onde a detecção de falhas perfeita é possível, enquanto a replicação máquina de estados permite aos clientes o envio de conjuntos de operações, que serão executadas sem contenção, desde que o problema de consenso tenha solução no modelo de sistema empregado. Este trabalho não considera o desempenho destas técnicas nem analisa os mecanismos empregados para tolerância a faltas Bizantinas.

Em [Goodson et al., 2004] é apresentado um serviço de armazenamento tolerante a faltas Bizantinas eficiente e escalável baseado em sistemas de quóruns. A solução apresentada neste artigo é comparada a uma implementação baseada em replicação máquina de estados [Castro and Liskov, 2002]. Os resultados demonstram que a solução baseada em quóruns oferece melhor desempenho (em termos de tempo de resposta) quando o número de faltas toleradas no sistema aumenta. Em nosso estudo, não avaliamos o aspecto da **escalabilidade no número de servidores** e nem consideramos detalhes de implementação de sistemas específicos, uma vez que os sistemas aqui comparados foram concretizados no mesmo arcabouço usando mecanismos semelhantes. Outro trabalho, apresentado em [Abd-El-Malek et al., 2005], introduz um modelo de replicação geral que se baseia apenas em sistemas de quóruns. Este modelo requer um número maior de réplicas ($n \geq 5f + 1$) e não garante a terminação de operações concorrentes. Mesmo com estas limitações, o trabalho demonstra que sua abordagem é mais escalável que um algoritmo eficiente de replicação máquina de estados [Castro and Liskov, 2002], i.e., é mostrado que o protótipo do modelo proposto mantém uma boa resposta em termos de requisições atendidas por unidade de tempo mesmo com o aumento do número de faltas toleradas no servidores. Apesar de demonstrar a escalabilidade dos sistemas de quóruns, esta análise sofre dos mesmos problemas daquela apresentada em [Goodson et al., 2004]: ela compara sistemas específicos, e não técnicas implementadas de forma similar.

7. Considerações Finais e Perspectivas Futuras

Este trabalho apresentou os resultados preliminares de uma comparação entre as duas principais técnicas para implementação de tolerância a faltas Bizantinas: replicação máquina de estados e sistemas de quóruns. Dados os resultados obtidos nos experimentos realizados, é possível derivar pelo menos três conclusões:

Ineficiência dos protocolos de quóruns: os protocolos de quóruns que requerem dados auto-verificáveis geralmente fazem uso de assinaturas digitais, que são computacional-

[5]Execução de um conjunto de operações de forma atômica.

mente dispendiosas. Devido a este fato, o protocolo de escrita do BFT-BC tem um desempenho muito inferior ao PAXOS em redes locais (onde a latência de comunicação é mínima). Além disso, o alto custo do processamento executado nos servidores torna este protocolo pouco escalável e suscetível a ataques de negação de serviço (ver Figura 4(b)). Estes comportamentos ressaltam a asserção de que criptografia assimétrica é o principal fator de impacto no desempenho em um protocolo quando executado em redes locais, e portanto deve ser evitada a todo custo.

Eficiência da replicação máquina de estados: os resultados dos experimentos realizados mostram que a replicação máquina de estados é bastante eficiente em casos sem falha, isto pode ser explicado por dois fatos: (1) o não uso de criptografia assimétrica nestes casos; e (2) a baixíssima latência e a previsibilidade das comunicações em redes locais, que tornam praticamente impossível a troca de *rounds* em casos sem falha. Além disso, a otimização utilizada para realização de leitura sem uso de ordem total é muito proveitosa em redes locais tendo em vista que, neste tipo de ambiente, a ordem total espontânea [Pedone and Schiper, 1998] mantém os servidores quase sempre no mesmo estado.

Escalabilidade da replicação máquina de estados: o gráfico da Figura 4 mostra que a latência desta abordagem aumenta de forma bastante "suave" com o aumento da concorrência no sistema. Este comportamento se deve à implementação de ordenação em lotes: o consenso para ordenação da mensagens é iniciado periodicamente, e não a cada mensagem recebida.

Estas conclusões vão contra o senso comum de que os protocolos para sistemas de quóruns são sempre eficientes e escaláveis: isto não se verifica quando consideramos baixa latência de comunicação, um número mínimo de servidores e execuções sem falha. Além disso, elas mostram que a replicação máquina de estados baseada no PAXOS é uma técnica viável e prática, mesmo sem grandes otimizações de baixo nível[6].

Entre as perspectivas futuras, podemos citar pelo menos três continuações possíveis para este trabalho: *(i.)* **Avaliação dos protocolos em redes de larga escala**; em um cenário deste tipo, espera-se que os sistemas de quóruns apresentem melhor desempenho, já que seus protocolos requerem menos trocas de mensagens e a maior latência de comunicação tende a diluir o custo da criptografia assimétrica; *(ii.)* **Protocolos de consenso descentralizados**; conforme demonstrado pelos experimentos, o desempenho do PAXOS decai sensivelmente quando ocorrem trocas de *rounds*. De fato, este comportamento é inerente a todos os protocolos baseados em coordenador. Uma possível extensão deste trabalho seria avaliar o desempenho geral de protocolos completamente distribuídos. Esta abordagem pode ser promissora uma vez que estudos recentes demonstram que estes protocolos apresentam um bom desempenho em redes locais, e este desempenho não sofre nenhuma degradação em caso de falhas [Moniz et al., 2006]; e *(iii.)* **Outros protocolos para sistemas de quóruns**; os experimentos mostram que as escritas do BFT-BC são ineficientes devido ao uso de assinaturas criptográficas (necessárias para tornar os dados armazenados auto-verificáveis). No entanto, existem outros protocolos para sistemas de quóruns que oferecem as mesmas garantias do BFT-BC ao custo de mais servidores e mensagens [Bazzi and Ding, 2004, Cachin and Tessaro, 2006]. Uma análise de desempenho destes protocolos seria, com certeza, uma interessante extensão ao presente trabalho. Além disso, protocolos com garantias mais fracas apresentam um melhor desempenho,

[6] Ao contrário da implementação apresentada em [Castro and Liskov, 2002], bastante otimizada e escrita na linguagem C, nosso serviço é concretizado como um protótipo escrito em Java.

porém não toleram clientes faltosos e implementam semânticas mais fracas. A avaliação destes protocolos pode nos dar algumas impressões sobre o quão custoso é tolerar clientes maliciosos e/ou implementar semântica atômica em protocolos para sistemas de quóruns.

Referências

Abd-El-Malek, M., Ganger, G., Goodson, G., Reiter, M., and Wylie, J. (2005). Fault-scalable Byzantine fault-tolerant services. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles - SOSP'05*, pages 59–74.

Bazzi, R. A. and Ding, Y. (2004). Non-skipping timestamps for Byzantine data storage systems. In *Proc. of 18th Int. Symposium on Distributed Computing, DISC 2004*, volume 3274 of *LNCS*, pages 405–419.

Cachin, C. and Tessaro, S. (2006). Optimal resilience for erasure-coded Byzantine distributed storage. In *Proceedings of the International Conference on Dependable Systems and Networks - DSN 2006*.

Castro, M. and Liskov, B. (2002). Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461.

Ekwall, R. and Schiper, A. (2005). Replication: Understanding the advantage of atomic broadcast over quorum systems. *Journal of Universal Computer Science*, 11(5):703–711.

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

Goodson, G. R., Wylie, J. J., Ganger, G. R., and Reiter, M. K. (2004). Efficient Byzantine-tolerant erasure-coded storage. In *Proceedings of International Conference on Dependable Systems and Networks - DSN 2004*.

Herlihy, M. (1991). Wait-free synchronization. *ACM Transactions on Programing Languages and Systems*, 13(1):124–149.

Lamport, L. (1986). On interprocess communication (part II). *Distributed Computing*, 1(1):203–213.

Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programing Languages and Systems*, 4(3):382–401.

Liskov, B. and Rodrigues, R. S. M. (2006). Tolerating byzantine faulty clients in a quorum system. In *The 26th IEEE International Conference on Distributed Computing Systems - ICDCS 2006*.

Malkhi, D. and Reiter, M. (1998). Byzantine quorum systems. *Distributed Computing*, 11(4):203–213.

Martin, J.-P. and Alvisi, L. (2005). Fast Byzantine consensus. In *Proceedings of the Dependable Systems and Networks - DSN-2005*.

Martin, J.-P., Alvisi, L., and Dahlin, M. (2002). Minimal Byzantine storage. In *Proceedings of the 16th International Symposium on Distributed Computing - DISC 2002*, volume 2508 of *LNCS*, pages 311–325.

Moniz, H., Neves, N. F., Correia, M., and Veríssimo, P. (2006). Randomized intrusion-tolerant asynchronous services. In *Proceedings of the International Conference on Dependable Systems and Networks - DSN 2006*.

Obelheiro, R. R., Bessani, A. N., and Lung, L. C. (2005). Analisando a viabilidade da implementação prática de sistemas tolerantes a intrusões. In *Anais do V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2005*.

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

Schneider, F. B. (1990). Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

Toueg, S. (1984). Randomized Byzantine agreements. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 163–178.

Urbán, P., Défago, X., and Schiper, A. (2001). Chasing the FLP impossibility in a LAN or how robust can a fault tolerant server be? In *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems - SRDS'01*, pages 190–193.

Urbán, P., Défago, X., and Schiper, A. (2002). Neko: A single environment to simulate and prototype distributed algorithms. *Journal of Information Science and Engineering*, 18(6):981–997.

Zhou, L., Schneider, F., and Van Rennesse, R. (2002). COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368.

Zielinski, P. (2004). Paxos at War. Technical Report UCAM-CL-TR-593, University of Cambridge Computer Laboratory, Cambridge, UK.