Avaliando o uso de Espaço de Tuplas como Alternativa para Implementação de Serviços Tolerantes a Faltas Bizantinas

Aldelir Fernando Luiz[1], **Lau Cheuk Lung**[2],
Alysson Neves Bessani[3]

[1]DAS - Departamento de Automação e Sistemas - UFSC

[2]INE - Departamento de Informática e Estatística - UFSC

[3]LaSIGE - Faculdade de Ciências da Universidade de Lisboa

aldelir@das.ufsc.br, lau.lung@inf.ufsc.br, bessani@di.fc.ul.pt

Abstract. *The State Machine Replication is one of the most interesting techniques to build reliable and safe services. This paper presents an experimental evaluation over a set of protocols for Byzantine Fault-Tolerant state machine replication that are practical. The evaluation is done in order to verify situations on that alternative coordination models are more attractive than message passing to implement Byzantine Fault-Tolerant services. For that, microbenchmarks and macrobenchmarks were done in order to compute latency and throughput measures.*

Resumo. *A replicação de máquina de estados é uma das técnicas mais usuais para a concepção de sistemas confiáveis e seguros. Este trabalho apresenta uma avaliação experimental acerca de um conjunto de protocolos para replicação de sistemas tolerantes a faltas bizantinas com fins práticos. O intuito desta avaliação é verificar situações em que modelos de comunicação alternativos ao tradicional modelo de passagem de mensagens tornam-se mais atraentes para a concretização de serviços tolerantes a faltas bizantinas. A avaliação foi realizada através de microbenchmarks e macrobenchmarks, computando-se as medidas de latência e throughput.*

1. Introdução

Um aspecto bastante interessante e favorável quanto ao conceito de sistemas distribuídos, é que ele proporciona um suporte inerente à redundância, um dos princípios fundamentais da tolerância a faltas. Além do mais, a combinação de redundância com mecanismos e técnicas adequadas permite implementar serviços capazes de atender aos requisitos de confiabilidade, integridade e disponibilidade, requisitos estes essenciais no que tange a confiança de funcionamento [Avizienis et al. 2004] do sistema de computação.

Independente da semântica de falhas admitida no contexto de um sistema distribuído, a redundância é um dos mecanismos mais usuais (e fundamentais) para a implementação de sistemas tolerantes a faltas, isto é, sistemas que visam garantir a continuidade e progressão do serviço/aplicação, mesmo que um determinado número de processos falhe em suas especificações. Em geral, a redundância é obtida por meio do emprego de técnicas de replicação, onde o modelo conhecido como *Replicação de Máquina de Estados* (RME) [Schneider 1990] é caracterizado como a abordagem mais empregada

no contexto de tolerância a faltas, dada sua capacidade de tolerar qualquer classe de faltas. Mais precisamente, a abordagem de RME consiste em uma técnica bem definida para a implementação de serviços deterministas a partir de um conjunto de servidores, de tal forma que é possível coordenar as interações dos clientes ao serviço (as réplicas), permitindo que as réplicas mantenham uma evolução sincronizada de seus estados. Para tanto, cada um dos servidores mantém um conjunto de variáveis de estado, que são modificadas na medida em que operações são emitidas sobre elas.

Para permitir a evolução sincronizada do conjunto de servidores, a abordagem RME admite um requisito conhecido como **determinismo de réplica** [Schneider 1990]. Este requisito estipula que réplicas partindo de um mesmo estado inicial e sujeitas à mesma seqüencia de operações, devem chegar ao mesmo estado final. Em termos práticos, a viabilização destes requisitos é obtida por intermédio de protocolos de *difusão atômica* [Défago et al. 2004], já que as propriedades intrínsecas a difusão atômica são suficientes para assegurar a consistência do estado das réplicas (acordo e ordem). Desta forma, como a evolução das réplicas ocorre de forma sincronizada, todas elas recebem as requisições, processam e enviam respostas aos clientes. As inconsistências verificadas nas respostas advindas das réplicas maliciosas são mascaradas a partir da aplicação de mecanismos de voto majoritário, onde é consolidada a entrega de um único resultado consistente ao cliente. A abordagem RME foi inicialmente definida como um mecanismos para prover tolerância à faltas de parada (*crash*) [Schneider 1990], sendo posteriormente estendida para a classe de faltas bizantinas [Castro and Liskov 1999].

Em se tratando de faltas bizantinas [Lamport et al. 1982], apesar de o conceito ter sido introduzido há quase trinta anos, durante alguns anos o interesse de investigação neste campo havia caído no esquecimento, tendo ressurgido com intensidade na última década a partir de um trabalho seminal publicado pelo MIT, denominado PBFT (*Practical Byzantine Fault-Tolerance*) [Castro and Liskov 1999]. Antes da era estabelecida pelo PBFT, as propostas de algoritmos tolerantes a faltas bizantinas eram dependentes de premissas muito fortes e de mecanismos criptográficos complexos, fazendo com que o custo de tais soluções fosse relativamente alto. Por esta razão, durante alguns anos houve especulações no sentido de que as soluções algorítmicas para BFT (*Byzantine Fault-Tolerance*) eram impraticáveis (não factíveis de implementação). E visando preencher esta lacuna, os trabalhos mais recentes no contexto de tolerância a faltas bizantinas (pós PBFT) têm aproveitado o ensejo do PBFT, tendo sua atenção voltada para a desmistificação dos conceitos prévios, através da proposição de soluções de cunho mais práticos e mais realistas (já que os trabalhos anteriores constituíram um imenso arcabouço teórico).

Por outro lado, a evolução das tecnologias de redes e de comunicação tem culminando no surgimento de novos modelos de sistemas distribuídos. Como conseqüência, a comunidade vem consolidando esforços no sentido de propor abstrações de comunicação alternativas ao **modelo de passagem de mensagens**, de modo a simplificar a tarefa de programação de serviços confiáveis para estes modelos. Dentre as diversas propostas, as mais interessantes do ponto de vista prático são o *Chubby* [Burrows 2006] (usado no *Google FileSystem*), o *ZooKeeper* [Apache 2009] (projeto da Apache) e o serviço de coordenação baseado em *Espaço de Tuplas* [Bessani et al. 2008]. O *Chubby* e o *ZooKeeper* são bastante similares a um sistema de arquivos distribuído, já o *Espaço de Tuplas* é baseado no **modelo de memória compartilhada distribuída** (DSM - *Distributed Shared Memory*). O elemento comum entre estes serviços de coordenação, é que a partir deles os

processos em um sistema distribuído podem coordenar suas atividades de forma simples e segura, segundo a semântica de falhas admitida no modelo do respectivo sistema.

Neste artigo investigamos a viabilidade prática de implementação de serviços confiáveis (i.e. tolerantes a faltas bizantinas) através do uso de um serviço de coordenação baseado em **espaço de tuplas**. Nosso interesse pelo **espaço de tuplas** advém da capacidade do mesmo em prover uma abstração de memória compartilhada confiável e segura, o qual fornece persistência de mensagens e capacidade de *logging* para o sistema, e, sobretudo, pode ser usado como suporte de comunicação e sincronização. Assim, os esforços de nosso grupo culminaram na definição da primeira arquitetura de suporte a implementação de serviços confiáveis baseado em espaço de tuplas, mais precisamente no sistema REPEATS [Luiz et al. 2008]. Este trabalho pode ser visto como uma extensão de [Luiz et al. 2008], tendo como propósito a verificação e análise no que concerne a viabilidade prática de implementação de serviços BFT em espaço de tuplas. Para tanto, o artigo descreve e analisa casos de experimentação a partir de *microbenchmarks* e *macrobenchmarks*, tendo como propósito a verificação e avaliação do modelo proposto em [Luiz et al. 2008] em diferentes cenários e condições de carga, com e sem a ocorrência de faltas/falhas.

2. Trabalhos Relacionados

Os trabalhos recentes em BFT têm produzido soluções baseadas em RME tolerante a faltas bizantinas, visando o fornecimento de serviços mais seguros e confiáveis. Tipicamente, as soluções algorítmicas para BFT admitem que no máximo $f \leq \lfloor \frac{n-1}{3} \rfloor$ membros de um conjunto $|S| = n$ podem desviar arbitrariamente de suas especificações de forma simultânea durante uma janela de vulnerabilidade do sistema (i.e. um período). As contribuições de muitos destes trabalhos estão em torno de protocolos para difusão atômica (*total order multicast*) tolerante a faltas bizantinas, no intuito de assegurar o modelo de consistência conhecido como **linearização** [Herlihy and Wing 1990], um requisito necessário para manter a corretude (*safety*) do serviço fornecido. O protocolo PBFT [Castro and Liskov 1999] é um dos trabalhos mais bem sucedidos em tolerância a faltas bizantinas em sistemas práticos, em vistas ao conjunto de otimizações nele implementadas, e além do fato de que grande parte das soluções para replicação com faltas bizantinas é derivada dele (i.e. [Yin et al. 2003, Kotla et al. 2007]). Contudo, em termos de custo, o PBFT é razoavelmente alto, pois, para tolerar um único nó faltoso ($f = 1$), são necessárias pelo menos quatro réplicas ($3f + 1$). E além do mais, na medida em que cresce o parâmetro f, isto é, se é desejado tolerar mais de um nó faltoso, o número de réplicas cresce consideravelmente (i.e. $f = 4 \rightarrow n = 13$).

Em vistas ao custo em termos de resistência do PBFT (restrições teóricas do acordo bizantino [Lamport et al. 1982]), bem como do custo computacional do referido algoritmo (complexidade de mensagens), dois trabalhos posteriores propuseram alternativas para aperfeiçoar as limitações do PBFT tanto em termos de resistência [Yin et al. 2003] (redução do número de réplicas), como de complexidade de mensagens em condições favoráveis [Kotla et al. 2007] (execuções sem falhas). O primeiro deles [Yin et al. 2003] introduziu uma arquitetura para replicação de sistemas, estimulando a separação das tarefas desempenhadas pelo PBFT em duas camadas distintas. A motivação para a separação destas tarefas, que são respectivamente "acordo e execução", se deu através da verificação de que $2f + 1$ réplicas são o suficiente para mascarar faltas bizanti-

nas [Schneider 1990], a despeito das $3f+1$ entidades necessárias para se chegar ao acordo bizantino [Lamport et al. 1982]. Com base nestes pressupostos, se verificou a possibilidade de construir serviços tolerantes a faltas bizantinas a partir do uso de dois conjuntos de servidores, sendo um para executar o protocolo de acordo bizantino, e, portanto requerendo $3f+1$ réplicas, e outro com $2f+1$ réplicas para implementar o serviço replicado (a aplicação).

Por outro lado, visando a redução da complexidade de mensagens e o número de passos de comunicação foi proposto o protocolo *Zyzzyva* [Kotla et al. 2007], o qual introduziu o conceito de execução "especulativa" no contexto de faltas bizantinas. A noção de especulação está ligada a possibilidade de executar uma requisição em um serviço replicado, sem a necessidade de realizar um acordo explícito entre as réplicas, isto é, ao invés de tentar convergir em uma decisão quanto à ordem de execução de uma requisição ao serviço, as réplicas tentam acatar uma ordem emitida por um líder (uma réplica) e, uma vez tomado o conhecimento, a requisição é executada na ordem estipulada. A idéia do *Zyzzyva* é a concepção de um protocolo de ordenação otimista (já que o PBFT é bastante pessimista), pois na prática se observa que na maioria das vezes as execuções são favoráveis (livres de falhas). A noção de especulação traz grandes benefícios em termos de desempenho para o sistema replicado. Todavia, este tipo de solução requer a capacidade de retroceder operações para assegurar a consistência do sistema, no caso da ocorrência de falhas. Isso ocorre pelo fato de que na ocorrência de falhas, as operações têm de ser retrocedidas para fins de convergência quanto a uma nova ordem de execução das requisições, por parte das réplicas (para assegurar o *safety*).

Na seção 6. é apresentada uma avaliação experimental acerca dos trabalhos descritos nesta seção, juntamente a um protocolo para replicação de serviços baseado em espaço de tuplas (ver seção 4.).

3. Fundamentos em Espaço de Tuplas

Conceitualmente, um espaço de tuplas pode ser visto como uma abstração de memória compartilhada, tendo como propósito facilitar as atividades de interação entre processos distribuídos [Gelernter 1985]. A origem do espaço de tuplas se deu no contexto da linguagem LINDA, a partir da definição de um modelo denominado **modelo de coordenação generativa** [Gelernter 1985]. Neste modelo o espaço de tuplas é usado como suporte de coordenação e comunicação para os processos participantes do sistema, onde é fornecida uma interface de acesso ao espaço, de modo a permitir o armazenamento e a recuperação (inserção, leitura e remoção) de estruturas de dados genéricas sob a forma de tuplas. Uma **tupla** $t = \langle f_1, f_2, ..., f_n \rangle$ consiste da composição de uma seqüência de campos, onde cada campo f_i pode ter três representações: um valor definido, um formal (variável) "?" ou ainda um símbolo especial "*". Um campo formal é usado para extrair conteúdos individuais dos campos de uma tupla, já o símbolo especial é usado para representar um campo sem valor, nem tipo definido. Uma tupla t onde todos os campos têm valores definidos é denominada **entrada**. Todavia, uma tupla cuja composição admite algum campo formal "?" e/ou um campo especial "*" é denominada **molde**, e é representada por $\bar{t}$. O espaço permite somente o armazenamento de tuplas e nunca de moldes, pois estes últimos são usados como argumentos para as operações de leitura sobre o espaço. A leitura e/ou remoção de tuplas é realizada com base na combinação dos moldes com as respectivas tuplas do espaço. Deste modo, dizemos que um molde $\bar{t}$ **combina** com uma entrada t se ambos têm

o mesmo número de campos e todos os campos com valores definidos de $\bar{t}$ contém os mesmos valores dos campos correspondentes em t. Por exemplo, uma tupla $\langle \text{“}SBRC'', 2010\rangle$ combina com os moldes $\langle \text{“}SBRC'', *\rangle$, $\langle *, 2010\rangle$ e $\langle *, *\rangle$, mas não com $\langle *, 2009\rangle$ (sendo '*' um campo não definido para o molde $\bar{t}$).

As manipulações ao espaço de tuplas são realizadas por meio de invocações de três operações [Gelernter 1985]: $out(t)$ para inserção de uma tupla t no espaço de tuplas; $in(\bar{t})$ para a retirada da tupla t, que combina com o molde $\bar{t}$, do espaço de tuplas; e $rd(\bar{t})$ para a leitura da tupla t (que combina com $\bar{t}$) sem retirá-la do espaço. As operações de leitura e remoção são não-deterministas, e sendo assim, caso exista um conjunto de tuplas que combine com o molde especificado, qualquer uma delas pode ser escolhida como resposta da operação. Também convém salientar que as operações *in* e *rd* são bloqueantes, e caso não exista nenhuma tupla t que corresponda ao molde $\bar{t}$ no espaço, o processo permanece bloqueado até que uma tupla que combine com o molde seja inserida para que a operação seja completada. Todavia, uma extensão típica do modelo provisiona variantes não bloqueantes das operações de leitura e remoção a partir das primitivas inp e rdp. Estas operações possuem o mesmo comportamento de suas versões bloqueantes, exceto pelo fato delas retornarem mesmo quando não existe nenhuma tupla que combine com o molde fornecido (elas retornam um valor booleano que sinaliza se uma tupla foi encontrada ou não).

Um aspecto fundamental do espaço de tuplas é o seu acesso associativo às tuplas, isto é, as tuplas não são acessadas por meio de um endereço ou algo do gênero, mas sim pelo seu conteúdo, similar ao que ocorre em bases de dados a partir da linguagem SQL. Além do mais, a despeito das operações básicas, algumas extensões do modelo generativo suportam uma operação de inserção condicional denominada *Conditional Atomic Swap*, que é denotada por $cas(\bar{t}, t)$ [Segall 1995]. Esta operação recebe como argumentos um molde $\bar{t}$ e uma entrada t, e para estes argumentos executa de forma indivisível o código: **if** $\neg rdp(\bar{t})$ **then** $out(t)$. A semântica da operação $cas(\bar{t}, t)$ denota que se a leitura do molde $\bar{t}$ falhar, então a inserção da tupla t é realizada no espaço, do contrário, a tupla que combina com o molde $\bar{t}$ é retornada ao processo que invocou a operação. A considerar os problemas fundamentais em sistemas distribuídos, como é o caso do problema de consenso, somente através da operação cas é possível resolvê-los por intermédio de um espaço de tuplas [Segall 1995].

4. REPEATS: Replicação com Faltas Bizantinas em Espaço de Tuplas

Um aspecto de fundamental importância para o entendimento de nossa contribuição advém do fato de que todas as soluções existentes para BFT (incluindo as apresentadas na seção 2.) são baseadas no modelo de comunicação por passagem de mensagens, modelo tradicional adotado em sistemas distribuídos. Todavia, a capacidade de resolução de problemas fundamentais de tolerância a faltas em sistemas distribuídos está intrinsecamente ligada à especificação de um modelo de sistema adequado. Em face deste fato, tem se verificado o interesse na proposição de abstrações de mais alto nível para prover facilidades na resolução de problemas de computação distribuída (i.e. consenso, acordo), e isso tem incorrido na investigação de modelos de comunicação alternativos ao modelo de passagem de mensagens, principalmente para a consolidação de aplicações distribuídas tolerantes a faltas (de diversas naturezas) [Burrows 2006, Bessani et al. 2008, Apache 2009, Clement et al. 2009]. Assim, a proposição destes modelos tem impulsionado também o

surgimento de contribuições para BFT com fins mais práticos, que admitem para tanto, estas abstrações no modelo de sistema.

Entretanto, apesar do recente interesse na utilização de abstrações de mais alto nível para a implementação de serviços tolerantes a faltas, a idéia foi introduzida há mais de uma década por [Felber et al. 1997], onde em um trabalho seminal os autores propuseram o uso do serviço de eventos do CORBA como mecanismo de coordenação e comunicação para a replicação de objetos distribuídos daquela arquitetura, embora a proposta não tenha se mostrado muito escalável nem tampouco prática. Por outro lado, o grande atrativo do uso de abstrações de comunicação alternativas, principalmente aquelas baseados em memória compartilhada, é que elas fornecem um suporte inerente para persistência e *logging* de mensagens, e, além de facilitarem em muito a especificação final dos sistemas, não requerem a definição de serviços adicionais.

Motivado por estas justificativas nosso grupo propôs o sistema REPEATS [Luiz et al. 2008], uma arquitetura para especificação e implementação de sistemas tolerantes a faltas bizantinas baseado puramente em espaço de tuplas. Até onde sabemos, nossos esforços foram pioneiros quanto à concretização de serviços replicados utilizando uma abstração de alto nível, neste caso o espaço de tuplas, já que o trabalho de [Felber et al. 1997] apenas tratava da especificação do sistema e não de sua implementação (que por sinal não chegou a ser realizada). A idéia básica do REPEATS é a concretização de um conjunto de algoritmos/protocolos para prover tolerância a faltas bizantinas, através de *replicação de máquina de estados*. Todavia, os algoritmos e protocolos da arquitetura foram especificados de forma a coordenar suas ações por meio de um espaço de tuplas confiável e seguro, de tal forma que todas as atividades pertinentes a coordenação e comunicação são realizadas através do espaço de tuplas.

Para melhor entendimento, o princípio básico de funcionamento do REPEATS é apresentado na figura 1. Nesta figura é ilustrada a interação de clientes com o serviço através do espaço, onde é verificado que tanto as requisições dos clientes como as respostas das réplicas são mapeadas em tuplas que contém os respectivos dados de invocação e resultados. Obviamente que as mesmas premissas e requisitos aplicados ao modelo RME (i.e acordo, ordem, sincronização etc), também o são para o sistema REPEATS, já que o mesmo é uma formalização de um novo modelo de replicação.

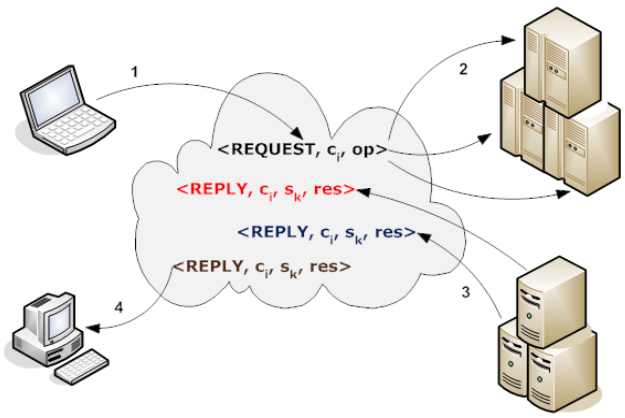


Figura 1. Princípio de Funcionamento da Replicação em Espaço de Tuplas

Em suma, o protocolo de replicação do REPEATS é composto por dois subprotocolos, que são respectivamente o protocolo de ordenação e o de *checkpointing*.

Adicionalmente são definidos mecanismos de suporte a replicação tais como *logging* estável, confiável e persistente, e coleta de lixo. O protocolo de ordenação é suportado por uma fila de mensagens persistentes (construída sobre o espaço de tuplas), de forma análoga ao protocolo de comunicação em grupo proposto para o sistema SINFONIA [Aguilera et al. 2007]. Além do mais, o REPEATS pode ser visto como a especificação da "separação de acordo e execução" [Yin et al. 2003] em espaço de tuplas, já que o algoritmo de ordenação é concretizado sobre o espaço de tuplas, constituindo assim uma camada de acordo. Deste modo, a execução dos serviços é em separado do protocolo de acordo. Devido a restrição de espaço, maiores detalhes a respeito da especificação e implementação da arquitetura REPEATS podem ser obtidos em [Luiz et al. 2008].

5. Modelo de Sistema

Para a concretização deste trabalho admitimos um modelo de sistema híbrido, dada a sensível diferença entre nossa proposta, cujos algoritmos são coordenados a partir de uma abstração de memória compartilhada que os demais trabalhos o fazem por meio de passagem de mensagens. Neste sentido, admitimos o uso de um conjunto arbitrário (não infinito) de clientes $C = \{c_1, c_2, ..., c_n\}$ que interagem com os serviços; para os sistemas PBFT [Castro and Liskov 1999] e *Zyzzyva* [Kotla et al. 2007] o conjunto de servidores é denotado por $S = \{s_1, s_2, ..., s_n\}$; para a arquitetura de separação de [Yin et al. 2003] o conjunto S é composto por dois subconjuntos, $A = \{a_1, a_2, ..., a_n\}$ para os servidores da camada de acordo, e $E = \{e_1, e_2, ..., e_n\}$ para os servidores da camada de execução; e por fim, no caso do sistema REPEATS os processos também estão divididos em dois subconjuntos, sendo $S_{ts} = \{s_{ts1}, s_{ts2}, ..., s_{tsn}\}$ para os servidores que implementam o espaço de tuplas, e $S_r = \{s_{r1}, s_{r2}, ..., s_{rn}\}$ para os servidores que implementam as réplicas do serviço.

Em relação às falhas, os processos estão sujeitos a faltas bizantinas [Lamport et al. 1982], os quais podem desviar arbitrariamente de suas especificações podendo parar, omitir envio ou entrega de mensagens, enviar respostas incorretas, entre outros. Contudo, cabe ressaltar que para os sistemas PBFT e *Zyzzyva* até $f \leq \lfloor \frac{n-1}{3} \rfloor$ servidores podem falhar simultaneamente; para a arquitetura de separação, não mais de $f_A \leq \lfloor \frac{n-1}{3} \rfloor$ **servidores de acordo** e $f_E \leq \lfloor \frac{n-1}{2} \rfloor$ **servidores de execução** podem falhar. Para o sistema REPEATS, até $f \leq \lfloor \frac{n-1}{2} \rfloor$ servidores podem falhar, requerendo, portanto, que não mais de $f \leq \lfloor \frac{n-1}{3} \rfloor$ réplicas do espaço de tuplas falhem para manter a correção do sistema. Não há um limite de número de faltas para os processos que representam os clientes dos sistemas.

Todos os sistemas avaliados mantêm as propriedades de correção no modelo de interação assíncrono (tempos desconhecidos). Todavia, para assegurar a progressão dos sistemas, todos admitem o modelo síncrono terminal (*eventually synchronous* [Dwork et al. 1988]), onde existem períodos de estabilização resultando em computações e comunicações terminalmente síncronas. Por fim, todos os sistemas admitem que as comunicações ocorrem por meio de canais ponto-a-ponto confiáveis (inclusive o REPEATS, em nível subjacente de comunicação com a abstração de espaço de tuplas).

6. Implementação, Avaliação e Resultados

A avaliação de desempenho de sistemas de computação distribuída pode ser realizada de forma analítica (teórica) ou empírica (pragmática) [Jain 1991]. A abordagem de avaliação

analítica concerne aos aspectos mais teóricos e tem o objetivo de medir a eficiência de um sistema através do estudo das complexidades algorítmicas. Por outro lado, a abordagem de avaliação empírica é baseada em amostras de dados, que são extraídas por meio de execuções dos algoritmos do sistema em ambiente real ou simulado. Nosso estudo foi concretizado sobre a abordagem de avaliação pragmática, em vistas as diferenças existentes nos modelos de sistema dos protocolos avaliados. Mais precisamente, o que impede uma avaliação teórica dos protocolos é o fato de que as soluções usuais para BFT coordenam as atividades dos algoritmos por meio de passagem de mensagens, diferente do sistema REPEATS que o faz por meio de memória compartilhada. Neste sentido, caso se optasse por uma avaliação analítica, esta não teria sentido algum, já que as métricas empregadas para a avaliação da complexidade de algoritmos distribuídos em memória compartilhada são diferentes daquelas usadas para medir a complexidade de algoritmos distribuídos baseados em trocas de mensagens [Attiya and Welch 2004]. E por esta razão, nossa avaliação foi somente realizada sob fundamentos pragmáticos e concretizada por meio de experimentações em ambiente real (avaliação empírica).

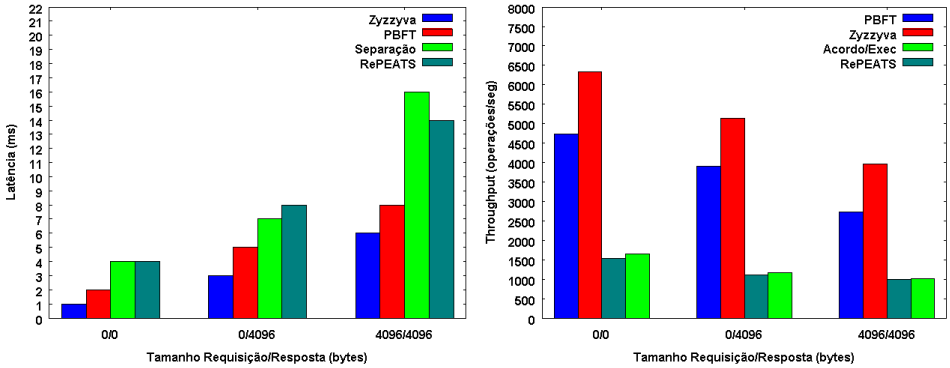
Em termos de implementação, todas as codificações foram efetuadas na linguagem Java, a partir do JDK 1.6.0_7 da SUN. A concretização dos canais de comunicação confiáveis e autenticados (conforme estipulado no modelo de sistema) ocorreu por meio do uso dos *socket channels* TCP da API NIO, juntamente com o uso de MACs (*Message Authentication Code*). A opção pela linguagem Java se deu em virtude da implementação do sistema de espaço de tuplas, o DEPSPACE [Bessani et al. 2008] estar disponível nesta linguagem. Já a opção pelo uso da NIO ocorreu em vistas às otimizações concretizadas nesta API. Para a realização da avaliação constituímos o ambiente de experimentação a partir de um conjunto de máquinas Dell Optiplex 755, todas com configuração homogênea em termos de *hardware* e de *software*. A composição do *hardware* foi a seguinte: 1 processador Intel®Core™2 Duo 2.33GHz, 2GB RAM e uma interface Ethernet Gigabit Intel 82566DM-2. Para o ambiente de *software* escolhemos o S.O SUSE Linux SLES 10 (*Kernel* 2.6.16.21-0.8-smp x86-64) equipado com a JVM-JIT IBM 1.6.0. Optamos pela JVM-JIT da IBM por ela apresentar melhor desempenho que a JVM Sun. O número de máquinas usadas para cada experimento foi variado de acordo com o número de faltas permitidas (parâmetro f). Além disso, o número mínimo de máquinas exigido para cada experimento variou também de acordo com o protocolo executado. Para os protocolos PBFT e *Zyzzyva* foram necessárias pelo menos $n_{AE} = 3f_{AE} + 1$ máquinas. Assim, estas n_{AE} foram usadas também (em experimentos distintos) para executar os servidores de acordo da "arquitetura de separação", bem como para as réplicas do espaço de tuplas confiável (requisito do DEPSPACE). Para a execução da camada de execução da "arquitetura de separação" e para as réplicas do serviço confiável implementado pelo sistema REPEATS foram usadas outras $n_E = 2f_E + 1$. Deste modo, tolerou-se f_{AE} e f_E réplicas faltosas de acordo com o experimento realizado.

As métricas adotadas para a avaliação foram a latência e o *throughput*, em virtude destas serem largamente usadas na avaliação de sistemas de computação, e pelo fato delas permitirem verificar de forma simplificada a eficiência do sistema [Jain 1991]. As medidas foram obtidas a partir da realização de *microbenchmarks* e *macrobenchmarks*, e em diversas condições de carga. As medidas de latência foram obtidas a partir da transmissão de um conjunto de mensagens, e da espera das respostas dos receptores (*round-trip*). Já as medidas de *throughput*, aqui representam a capacidade máxima de processamento de

requisições por unidade de tempo, e, para a obtenção destes valores nos baseamos no tempo de processamento das requisições, desde o envio até a recepção de todas as respostas advindas das réplicas. A motivação para a avaliação por *microbenchmarks* foi a possibilidade de analisar o custo (em unidade de tempo) dos algoritmos de ordenação, sem a influência de uma aplicação/serviço. A idéia por trás do *microbenchmark* foi verificar o impacto que o protocolo de ordenação/replicação causou no sistema confiável. Além do mais, um aspecto de grande relevância na avaliação de desempenho é obter o conhecimento do comportamento do protocolo para diferentes configurações e níveis de atividades, para verificar a escalabilidade em relação ao número de réplicas e tamanho das mensagens. Para tanto, a execução dos protocolos avaliados foi realizada por meio de um serviço sem estado e com operações nulas, variando os tamanhos das mensagens de requisição e respostas em 0KB e 4KB.

6.1. Avaliação de Desempenho no Cenário com Execuções Favoráveis

A primeira experimentação foi realizada com base na execução dos protocolos avaliados em condições normais, isto é, sem a presença de faltas. Em ambos os experimentos da figura 2 foram executadas 10000 operações oriundas de 30 clientes concorrentes, sendo que para a tabulação dos resultados foi desprezado 1% dos valores com maior desvio, conforme percebido nas execuções. Os resultados em termos de latência são reportados na figura 2(a), onde estes foram obtidos a partir de execuções com diferentes condições de carga, em função dos tamanhos das requisições e respostas. Os valores reportados compreendem o tempo médio de processamento de requisições, conforme observado nas execuções. Os resultados (figura 2(a)) demonstram que a latência das operações do sistema REPEATS é a maior de todas, mas muito próximo ao da arquitetura de separação de [Yin et al. 2003]. Cabe ressaltar que estes resultados já eram esperados, pois o sistema REPEATS e a arquitetura de [Yin et al. 2003] compartilham as mesmas características funcionais, e, portanto, requerem mais passos de comunicação do que os demais protocolos.



(a) Latência em operações normais. (b) *Throughput* em operações normais.

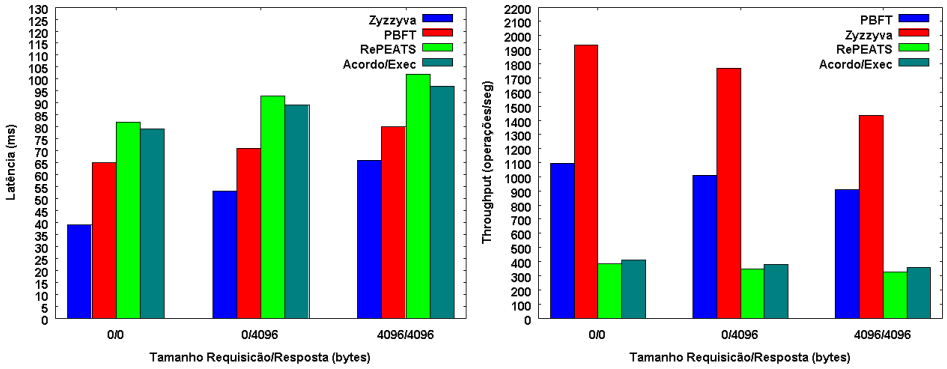
Figura 2. Desempenho de Operações Normais.

O razão para que os resultados de [Luiz et al. 2008] e [Yin et al. 2003] serem muito próximos se fundamenta no fato do DEPSPACE [Bessani et al. 2008] ser baseado no algoritmo de replicação PAXOS Bizantino, que por sua vez é equivalente ao PBFT (usado para o acordo em [Yin et al. 2003]). Além disso, a latência do REPEATS em relação a [Yin et al. 2003] é pior devido ao custo adicional de acesso ao espaço de tuplas.

Para as medidas de desempenho do *throughput* (vazão), os resultados são apresentados na figura 2(b). A verificação dos resultados mostra que o REPEATS tem o menor desempenho em relação aos protocolos avaliados, no entanto, tendo os resultados próximos aos reportados para a arquitetura de "separação do acordo e execução". Por outro lado se analisarmos o *throughput* em relação ao tamanho das requisições e respostas, se verifica que o sistema REPEATS apresenta escalabilidade aceitável, tendo em vista que o decréscimo do número de operações/segundo é pequeno quando se aumenta o tamanho das requisições e respostas. Estes resultados também mostram que o sistema tem escalabilidade razoável quando se aumenta o tamanho das requisições e respostas, pois o acréscimo na latência é menos de duas vezes. O caso particular das requisições e respostas com 4096 bytes é considerável, visto que o modelo adotado requer um passo de comunicação a mais para o envio da requisição ordenada aos servidores de execução, e isso incorre em um ônus ao sistema quando o tamanho das mensagens é relativamente grande.

6.2. Avaliação de Desempenho no Cenário com Execuções Faltosas

A fim de avaliar os protocolos com comportamentos de falhas, foram realizados os mesmos experimentos da seção anterior, porém, com a injeção de faltas. Assim, foram introduzidas faltas nos algoritmos PBFT, *Zyzzyva*, na arquitetura de separação e também no sistema REPEATS. Mais precisamente, as simulações contemplaram a injeção de faltas nas réplicas que exerceram a função de líder (por este representar o pior caso), de forma a induzir o protocolo a não obter o acordo na primeira rodada da execução (causando um *delay* em razão da troca de líder). Para o REPEATS e arquitetura de separação, as faltas foram injetadas na camada de acordo, já que as faltas da camada de execução pouco influenciam no desempenho do sistema, por estas serem mascaradas.

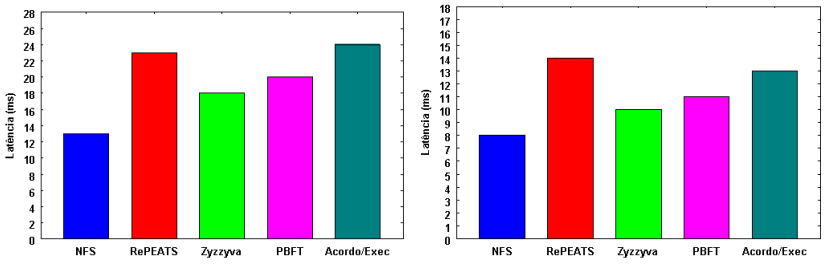


(a) Latência em operações com faltas. (b) *Throughput* em operações com faltas.

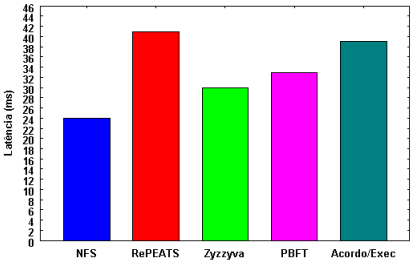
Figura 3. Desempenho de Operações com Faltas.

No primeiro momento foi avaliada a latência da execução de operações nulas nos protocolos avaliados. O resultado é equivalente ao do experimento anterior, no sentido de que tanto o REPEATS como a arquitetura de [Yin et al. 2003] apresentaram maior latência em relação aos demais protocolos, pois os valores reportados contemplam o tempo de execução dos protocolos de acordo (PAXOS e BFT) acrescido do tempo necessário para o envio das mensagens ordenadas aos servidores de execução. Como pode ser observado na figura 3(a), apesar do desempenho de nossa solução não ser o melhor, neste quesito o REPEATS é levemente inferior a arquitetura de [Yin et al. 2003]. Isso ocorre devido ao controle de acesso adicional do espaço de tuplas. Nos resultados reportados para o *throughput* (figura 3(b)) se verifica que todos os protocolos tiveram seu desempenho comprometido na

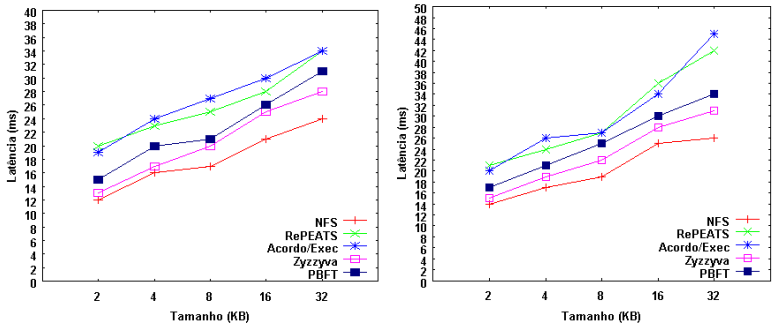
ocorrência de faltas, já que nestas situações os mesmos têm de executar serviços e rotinas adicionais para retornar as condições normais e garantir a progressão do sistema.



(a) Criação de arquivos. (b) Exclusão de arquivos.



(c) Listagem de diretórios.



(d) Leitura em arquivos. (e) Escrita em arquivos.

Figura 4. Latência de operações sobre objetos nos serviços NFS: NFS nativo (API *java.io* + cliente NFS Linux), REPEATS, Arquitetura de Separação, Zyzzyva e PBFT.

6.3. Avaliação da Implementação de Serviços NFS Tolerantes a Faltas Bizantinas

Conforme citado, a construção do *microbenchmark* é útil para verificar o impacto de uma solução sobre um serviço replicado. No entanto, um *macrobenchmark* se torna essencial a partir do momento em que se deseja avaliar o desempenho de uma solução sobre uma aplicação real. E para este fim, avaliamos a implementação de um serviço NFS replicado, tendo como base todos os protocolos avaliados. Esta avaliação foi realizada nos mesmos moldes dos trabalhos relacionados [Castro and Liskov 1999, Yin et al. 2003, Kotla et al. 2007]. Para tanto, nossa implementação foi inspirada nas especificações do WebNFS [Callaghan 1996a, Callaghan 1996b], que é uma extensão da especificação NFS, que permite algumas facilidades no acesso aos objetos remotos (arquivos, diretórios, *links*). Com o WebNFS as aplicações podem obter o acesso aos objetos remotos sem a necessidade de interação com o sistema operacional (não é necessário montar o sistema de arquivos remoto no *Virtual File System* do sistema operacional do cliente). O acesso aos objetos é feito através de uma API instanciada pela aplicação cliente.

Os resultados do *macrobenchmark* foram frutos da implementação de serviços NFS sobre cada um dos protocolos avaliados, além do NFS não replicado. Os experimentos compreenderam a execução de algumas operações comuns em arquivos, de modo a permitir a verificação da latência para cada operação. As operações analisadas sobre o NFS foram: (1) criação de arquivos e diretórios (figura 4(a)), (2) exclusão de arquivos e diretórios (figura 4(b)), (3) listagem do conteúdo de diretórios com 100 objetos recursivos (figura 4(c)), (4) escrita de 2k, 4k, 8k, 16k e 32k em arquivos remotos (figura 4(d)), e (5) leitura dos dados de arquivos com 2k, 4k, 8k, 16k e 32k (figura 4(e)). Para a avaliação foram realizadas 10000 execuções de cada operação, onde a latência reportada compreendeu o tempo médio para a execução da operação em questão, excluindo-se 1% dos valores com maior desvio.

A partir dos gráficos das figuras 4(a) e 4(b) se observa o que era esperado, tanto o sistema REPEATS como a arquitetura de [Yin et al. 2003] apresentaram os piores desempenhos. Contudo, mais uma vez é digno de nota que isso ocorre em virtude do modelo de replicação adotado, que requer mais um passo de comunicação. Se analisarmos de outro prisma (desconsiderando os outros trabalhos), podemos verificar que o custo adicional do uso do sistema REPEATS incorre em torno de 30 a 60% em relação ao serviço NFS não replicado (i.e não tolerante a faltas bizantinas). Este custo adicional se deve basicamente a latência extra de acesso ao **espaço de tuplas** durante a execução do protocolo de ordenação de requisições, mais precisamente pelo controle de acesso ao espaço (requisito para tolerar faltas bizantinas no REPEATS [Luiz et al. 2008]). Este custo é moderado se considerarmos os benefícios em termos de confiabilidade e segurança que o REPEATS oferece. Também é digno de nota o fato da latência apresentar um crescimento leve quando o tamanho da resposta do serviço é aumentado. Isto se deve ao fato das respostas serem enviadas diretamente aos clientes e não passarem pelo espaço de tuplas (uma otimização).

6.4. Avaliação das Propriedades dos Protocolos Tolerantes a Faltas Bizantinas

Uma última avaliação, porém, tão importante quanto às demais foi realizada sobre algumas propriedades dos protocolos BFT avaliados. Esta avaliação tem enfoque em três aspectos que os protocolos de replicação BFT devem se ater para tornar suas soluções algorítmicas mais práticas (factíveis de implementação), os quais são: custo em termos de número de réplicas; número de réplicas necessárias para manter o serviço, a despeito do mínimo requerido em replicação com faltas bizantinas ($2f + 1$ [Schneider 1990]); quantidade de serviços operando sobre uma única camada de acordo.

Tabela 1. Comparação das Propriedades dos Protocolos Avaliados

	REPEATS	**Acordo/Execução**	**PBFT**	**Zyzzyva**
Total de réplicas	$2f_e + 1 + \|S_{ts}\|$	$(3f_a + 1) + (2f_e + 1)$	$3f + 1$	$3f + 1$
Réplicas do serviço	$2f_e + 1$	$2f_e + 1$	$3f + 1$	$3f + 1$
Serviços/Camada acordo	N	$N*$	1	1

Os dados da tabela 1 refletem uma perspectiva quanto à adoção das soluções para BFT avaliadas, com vistas ao custo total de replicação. A primeira linha da tabela se refere ao número total de máquinas a serem usadas para construir um sistema tolerante a f faltas bizantinas, em cada um dos sistemas avaliados. A importância da redução do número de réplicas é crucial para a construção de um serviço, uma vez que isso implica também, no custo operacional do sistema (armazenamento, disco, etc.). Neste quesito

o sistema REPEATS se sobressai em relação aos demais por requerer o menor número de réplicas na implementação de serviços confiáveis, ainda que sejam necessárias as S_{ts} réplicas do espaço de tuplas que podem ser compartilhadas por diferentes serviços replicados e outras aplicações. Uma segunda avaliação é realizada com base no número de réplicas necessárias para manter o estado da aplicação, também tomando em conta o valor estipulado em [Schneider 1990]. Este número indica que tanto o sistema REPEATS como a arquitetura de [Yin et al. 2003] se destacam pelo fato deles compartilharem a mesma otimização que é o uso de serviços de acordo e execução separados. Esta otimização não se aplica a implementação original dos protocolos PBFT nem tampouco do *Zyzzyva*, já que nestes as mesmas réplicas executam o acordo e a aplicação. Por fim, a última linha da tabela nos mostra que tanto o sistema REPEATS como a arquitetura de [Yin et al. 2003] permitem o uso do mesmo serviço de acordo para implementar diversas aplicações confiáveis, a despeito dos demais trabalhos que não fornecem este tipo de otimização. No entanto, um fator favorável quanto ao sistema REPEATS é que o mesmo usa um serviço de coordenação genérico fornecido pelo espaço de tuplas para a camada de acordo. Desta forma, uma mesma instância do serviço de coordenação pode ser usada para outras aplicações e serviços que não dizem respeito ao sistema REPEATS, e assim tornando-o bastante atrativo em relação aos demais.

7. Conclusões

Neste trabalho apresentamos um estudo comparativo de um conjunto de protocolos para replicação com faltas bizantinas, com vistas à verificação da viabilidade de uso de uma abstração de memória compartilhada (espaço de tuplas) para a concepção de aplicações tolerantes a faltas bizantinas. A partir do presente estudo pudemos verificar que apesar do desempenho verificado, o uso extensivo do espaço de tuplas é interessante do ponto de vista prático, pois seu conjunto reduzido de operações permite especificar e implementar de forma bastante simples, sistemas distribuídos confiáveis e seguros, a despeito da complexidade do modelo de passagem de mensagens. Apesar da avaliação de desempenho nos mostrar que o espaço de tuplas implica em um incremento do custo de replicação (de latência) em relação aos demais trabalhos avaliados, o uso desta abstração se torna atraente, quando consideramos o incremento no número de faltas (parâmetro f) para um serviço.

Além disso, a única comparação real e plausível para a implementação de serviços em espaço de tuplas, é em relação ao trabalho que define a separação do acordo e execução [Yin et al. 2003], visto que ambos compartilham do mesmo princípio de funcionamento. Assim, os valores apresentados para os protocolos PBFT e *Zyzzyva* são apenas usados como referência, já que na prática eles sempre terão desempenho superior aos demais protocolos, em vistas ao número de passos de comunicação adicionais requeridos pelos sistemas que provêem separação das entidades de acordo e execução. Por outro lado, se avaliarmos o custo total de replicação em termos de número de máquinas/réplicas necessárias, se verifica que o uso do espaço de tuplas se torna mais atraente em razão do mesmo representar algo mais genérico que uma simples camada de acordo (conforme proposto em [Yin et al. 2003]). Assim, o mesmo espaço pode ser usado por diferentes aplicações, com diferentes propósitos (i.e. uma mesma instância do espaço).

Referências

Aguilera, M. K., Merchant, A., Shah, M., Veitch, A., and Karamanolis, C. (2007). Sinfonia: a new paradigm for building scalable distributed systems. In *SOSP '07: Proceedings of 21st ACM SIGOPS/Symposium on Operating Systems Principles*, pages 159–174, New York, NY, USA. ACM.

Apache, S. F. (2009). ZooKeeper Coordination Service. *http://hadoop.apache.org/zookeeper/*.

Attiya, H. and Welch, J. (2004). *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, 2nd edition.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Bessani, A. N., Alchieri, E. P., Correia, M., and Fraga, J. S. (2008). Depspace: a byzantine fault-tolerant coordination service. In *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 163–176, New York, NY, USA. ACM.

Burrows, M. (2006). The Chubby lock service for loosely-coupled distributed systems. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350, Berkeley, CA, USA. USENIX Association.

Callaghan, B. (1996a). WebNFS Client Specification (RFC 2054). IETF Request For Comments.

Callaghan, B. (1996b). WebNFS Server Specification (RFC 2055). IETF Request For Comments.

Castro, M. and Liskov, B. (1999). Practical Byzantine fault tolerance. In *OSDI '99: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pages 173–186. USENIX Association.

Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., and Riche, T. (2009). Upright cluster services. In *SOSP '09: Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pages 277–290, New York, NY, USA. ACM.

Défago, X., Schiper, A., and Urbán, P. (2004). Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421.

Dwork, C., Lynch, N. A., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–322.

Felber, P., Guerraoui, R., and Schiper, A. (1997). Replicating objects using the corba event service? In *Proceedings of the 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 14–19.

Gelernter, D. (1985). Generative communication in Linda. *ACM Transactions on Programing Languages and Systems*, 7(1):80–112.

Herlihy, M. and Wing, J. M. (1990). Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programing Languages and Systems*, 12(3):463–492.

Jain, R. K. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons.

Kotla, R., Alvisi, L., Dahlin, M., Clement, A., and Wong, E. L. (2007). Zyzzyva: speculative byzantine fault tolerance. In *SOSP '07: Proceedings of 21st ACM SIGOPS/Symposium on Operating Systems Principles*, pages 45–58, New York, NY, USA. ACM.

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

Luiz, A. F., Bessani, A. N., Lung, L. C., and Filgueiras, T. (2008). REPEATS: Uma arquitetura para replicação tolerante a faltas bizantinas baseada em espaço de tuplas. In *Anais do XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC.

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

Segall, E. J. (1995). Resilient distributed objects: Basic results and applications to shared spaces. In *SPDP '95: Proceedings of the 7th Symposium on Parallel and Distributed Processing*, pages 320–327.

Yin, J., Martin, J.-P., Venkataramani, A., Alvisi, L., and Dahlin, M. (2003). Separating agreement from execution for Byzantine fault tolerant services. In *SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 253–267.