RIST: Um *Middleware* para Replicação e Interoperabilidade de Sistemas Transacionais

Aldelir Fernando Luiz[1], Lau Cheuk Lung[1,2], Miguel Correia[3]

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

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

[3]Instituto Superior Técnico / INESC-ID - Portugal

aldelir@das.ufsc.br, lau.lung@inf.ufsc.br, miguel.p.correia@ist.utl.pt

Resumo. *Neste trabalho apresentamos o middleware RIST, uma solução para replicação de bases de dados transacionais, que além de encapsular os requisitos necessários para a interoperabilidade em ambientes heterogêneos é tolerante a faltas Bizantinas e totalmente distribuído. O RIST omite toda a complexidade do cliente, de forma que é apresentada ao mesmo uma única base de dados consistente. Além disso, a aplicação cliente não necessita de modificações, já que o RIST segue as especificações do JDBC e pode operar sobre sistemas de gerência de base de dados off-the-shelf proprietários e/ou de código aberto.*

Abstract. *This paper presents the middleware RIST, a solution for transactional database replication, which encapsulates the requirements for interoperability in heterogeneous environments. RIST is designed for Byzantine Fault Tolerance of a distributed database system. RIST omits all complexities from clients, in a transparent way where the database can be handled in a centralized fashion. Furthermore, the client application do not need any modification because RIST is compatible to the JDBC and can operates over owner or open source off-the-shelf database management systems.*

1. Introdução

O presente trabalho se insere no desafio 5 **(Desenvolvimento tecnológico de qualidade: sistemas disponíveis, corretos, seguros, escaláveis, persistentes e ubíquos)**, pois visa à busca de novas soluções tecnológicas, de baixo custo, em termos de modelos e arquiteturas, para um sistema de processamento de transações que dê suporte ao desenvolvimento de aplicações com alta disponibilidade, corretas e seguras mesmo em presença de ataques de segurança e intrusões [Lung 2009][1]. Os sistemas de processamento de transações podem ser considerados o cerne dos sistemas de informação que dão suporte as diversas áreas do âmbito socioeconômico. Em todas as áreas de conhecimento onde sistemas de informação são usados como ferramentas de apoio ao negócio (sistemas de gerenciamento de dados, sistemas de telecomunicação, sistemas de controle industrial, comércio eletrônico e outros) existe uma grande dependência quanto à confiabilidade e tempestividade dos dados manipulados por estes sistemas. Assim, é importante que os sistemas de gerenciamento de dados (e transações) sejam capazes de prover meios que permitam sua operação de forma contínua e consistente, a despeito de fatores externos (p.ex. ataques de segurança) que possam interferir no correto funcionamento do sistema. Desta forma, a replicação de dados [Gray et al. 1996]

[1]http://www.gd2.ufam.edu.br/

torna-se peça fundamental dentro do contexto de sistemas de gerência de bases de dados (SGBD).

Apesar dos comentários tecidos por [Gray et al. 1996] a respeito dos perigos encontrados na replicação de dados, nos últimos anos muitos trabalhos foram propostos no intuito de desmistificar os conceitos previamente definidos. A razão para isto decorre do fato de que a replicação de dados é um meio bastante factível para melhorar o desempenho e prover maior disponibilidade em sistemas de processamento de transações (ex. bases de dados). Por outro lado, quando os dados são residentes em mais de um sítio (seja replicado ou distribuído), o processo de confirmação da transação se torna mais custoso, pois uma única fase de confirmação é insuficiente para assegurar as propriedades **ACID** (**A**tomicidade, **C**onsistência, **I**solamento e **D**urabilidade) da transação [Bernstein et al. 1987]. Um método bastante conhecido para prover a confirmação de transações distribuídas e/ou replicadas é o *Two-Phase Commit* [Skeen 1981], em que o processo de confirmação é feito em duas etapas, que são a preparação e a confirmação.

A replicação de dados representa ainda, um desafio em se tratando de integração, interoperabilidade e consistência para os sistemas de gerenciamento de bases de dados. Assim, novas abordagens para replicação de bases de dados têm sido propostas na literatura, e algumas destas têm sido construídas sobre abstrações de comunicação em grupo [Schiper and Raynal 1996], de forma a prover soluções de cunho mais prático. Além disso, os protocolos e algoritmos que compõem estas soluções têm sido capazes de obter um elevado grau de consistência [Pedone et al. 2003, Cecchet et al. 2004, Vandiver et al. 2007] sem despender grande sobrecarga adicional, como observado em soluções de replicação de bases de dados usuais [Bernstein et al. 1987, Gray et al. 1996]. Neste cenário, primitivas de comunicação em grupo [Schiper and Raynal 1996] permitem especificar e implementar soluções para replicação mais eficientes, além de simplificar a adaptação de técnicas e mecanismos de tolerância a faltas [Birman and van Renesse 2005], visando tornar a base de dados resistente a alguns tipos de falhas (ex. parada, omissão, arbitraria).

Uma questão pertinente em termos de replicação de dados discorre a respeito da robustez das aplicações. Neste sentido, a abordagem de replicação ativa (ou de máquina de estados) [Schneider 1990] tem se mostrado como um elemento essencial para prover este quesito. Todavia, quando aplicada no contexto de sistemas de processamento de transações, é requerido, *a priori*, que as réplicas acordem acerca de uma ordem total para a execução das transações, para que o critério de serialização seja atendido. Este problema é um fator impeditivo, por razões de desempenho, para que a abordagem de replicação ativa pura seja de viável adoção em sistemas de processamento de transações. Em face deste fato, o trabalho de [Pedone et al. 2003] concretizou a adaptação da técnica de replicação ativa para o contexto de sistemas transacionais, o que deu origem a técnica conhecida na literatura como **Replicação de Máquina de Estados para Bases de Dados** (*Database State Machine Replication - DSMR*).

Neste trabalho apresentamos o projeto e implementação do RIST, uma solução de *middleware* para replicação de bases de dados transacionais, que consiste na concretização da técnica DSMR sob o modelo de faltas Bizantinas. Além disso, o RIST encapsula os requisitos necessários para permitir a interoperabilidade entre diferentes bases de dados, em um único ambiente replicado (e heterogêneo). O RIST mantém o controle das transações totalmente distribuído e omite toda a complexidade em termos de programação do cliente, de forma que é apresentada ao mesmo uma única base de dados consistente. Ademais, a

aplicação cliente não necessita de modificações, já que o *middleware* segue as especificações do **JDBC** [Fisher et al. 2003], provendo assim, o acesso ao ambiente confiável de forma totalmente transparente para a aplicação. O RIST pode operar sobre sistemas de gerência de base de dados *off-the-shelf* proprietário e/ou de código aberto, desde que eles atendam a alguns requisitos necessários ao funcionamento do protocolo (p.ex. consistência serializável), e, sobretudo, sem implicar em modificações nos sistemas de gerência de base de dados.

2. Conceitos em Replicação de Dados

Conforme mencionado, a replicação de dados é um aspecto de fundamental importância no contexto de sistemas transacionais (SGBDs). Esta importância advém do fato de que a replicação permite manter a disponibilidade do sistema em caso de falhas, e, sobretudo, oferece maior escalabilidade em face ao fato de possibilitar a execução em paralelo de transações em diferentes réplicas. Entretanto, o principal problema encontrado na replicação de dados está relacionado ao processo de convergência das réplicas, isto é, a forma pela qual as réplicas convergem para o mesmo estado, após a confirmação de uma transação, mais precisamente na forma como as operações devem ser propagadas às réplicas do sistema.

Em [Gray et al. 1996] é apresentada uma taxonomia para protocolos de replicação de dados, realizada por meio de dois parâmetros: (*i*) a forma pela qual as alterações são propagadas às réplicas; (*ii*) quem tem o privilégio de efetuar a propagação. Na abordagem de replicação passiva [Budhiraja et al. 1993], uma das réplicas do grupo é eleita líder e passa a ser a responsável por receber as requisições dos clientes e enviar as respostas das operações, além de ser a encarregada de propagar as atualizações às demais réplicas. Por sua vez, a abordagem de replicação ativa [Schneider 1990] é uma técnica descentralizada onde todas as réplicas recebem, executam e respondem as requisições, tendo como único requisito o determinismo. Para tanto, as réplicas partem de um mesmo estado inicial e executam a mesma sequência de operações, assim, é possível produzir os mesmos resultados.

Além das técnicas de replicação mencionadas, outra abordagem conhecida como baseada em certificação [Kung and Robinson 1981] é bastante aceita como forma de replicação de dados, pois permite que as transações sejam executadas de forma otimista em uma única réplica, onde a convergência é realizada apenas no momento da confirmação da transação através de um procedimento de coordenação e certificação, que é executado a fim de convergir o sistema a um estado global consistente. Assim, quando a aplicação solicita a confirmação da transação, é necessário que as operações da transação, bem como RS e WS (*Read-Set* e *Write-Set*)[2] sejam propagados através de um protocolo de difusão com ordem total [Défago et al. 2004]. Como resultado, quando a mensagem de propagação é entregue, todas as réplicas obtém a mesma visão do histórico de transações entregues e confirmadas, e com isso podem detectar possíveis conflitos com transações concorrentes da mesma maneira. Este processo de coordenação e certificação é necessário para que possa ser estabelecida uma ordem global para a confirmação das transações concorrentes, e assim, manter o sistema consistente.

3. Trabalhos Relacionados

Uma verificação na literatura nos mostra alguns trabalhos que tratam de replicação de bases de dados em ambientes heterogêneos [Cecchet et al. 2004, Vandiver et al. 2007] e outros que tratam da especificação de modelos de replicação para ambientes de bases

[2]RS e WS são os conjuntos de itens de dados afetados pelas operações de leitura e escrita da transação, respectivamente

de dados tolerantes e faltas Bizantinas [Pedone et al. 2011, Garcia et al. 2011]. Dentre eles, os mais próximos de nossa proposta são o C-JDBC [Cecchet et al. 2004], o HRDB [Vandiver et al. 2007], o Byzantium [Garcia et al. 2011] e o BFT-*Deferred Update* [Pedone et al. 2011], apesar do C-JDBC [Cecchet et al. 2004] não tolerar faltas Bizantinas [Lamport et al. 1982]. O C-JDBC define uma solução de *middleware* completa para replicação de bases de dados heterogêneas, e permite a criação de *clusters* de bases de dados para o uso em aplicações baseadas no JDBC. Apesar da capacidade de operar sobre múltiplas plataformas de bases de dados, o C-JDBC não provê suporte a ambientes sujeitos a faltas Bizantinas, mas apenas de parada (*crash*). As únicas soluções para replicação de bases de dados aptas a operar em ambientes mais severos tais como os sujeitos a faltas Bizantinas são o HRDB, o Byzantium e o BFT-*Deferred Update*. Todavia, apesar de o HRDB prover os requisitos de interoperabilidade para ambientes heterogêneos, ele é limitado no sentido de que o protocolo depende de um elemento central (que considera confiável) para conduzir o sistema [Vandiver et al. 2007], e assim, este coordenador não pode falhar durante as janelas de vulnerabilidade do sistema.

Por outro lado, Byzantium [Garcia et al. 2011] é uma solução que assegura o critério de consistência *snapshot isolation* em ambientes sujeitos a faltas Bizantinas, o qual é mais fraco do que o modelo de consistência ideal para o modelo de transação, o serializável. Além do *snapshot isolation* não garantir a consistência forte dos dados, neste modelo as transações observam apenas o instante da base de dados correspondente ao da última transação confirmada no sistema. Ademais, no *snapshot isolation* as transações concorrentes são capazes de ser confirmadas apenas se elas não modificaram os mesmos itens de dados, o que não ocorre no modelo serializável, modelo assegurado em nosso *middleware*. Por fim, a solução BFT-*Deferred Update* [Pedone et al. 2011] especifica um protocolo de replicação para bases de dados, que é tolerante a faltas Bizantinas, mas assume apenas réplicas maliciosas e não clientes. Assim, o protocolo considera os clientes "confiáveis", simplificando, portanto, o processo de confirmação das transações. O protocolo de replicação especificado para nosso *middleware* apresenta algumas similaridades em relação ao BFT-*Deferred Update*, no sentido de que adotamos o mesmo modelo de propagação de operações, que é o clássico *deferred update* [Bernstein et al. 1987]. Todavia, é lícito mencionar que nosso protocolo tolera faltas Bizantinas oriundas tanto das réplicas como dos clientes.

4. Arquitetura RIST - Replicação e Interoperabilidade em Sistemas Transacionais

O RIST consiste em uma arquitetura baseada em *middleware*, que provê um suporte para replicação de bases de dados heterogêneas em ambientes sujeitos a faltas Bizantinas. A concretização do RIST se dá através da implementação de um protocolo de replicação tolerante a faltas Bizantinas, a partir das especificações da API JDBC™, API esta que permite o acesso virtual a qualquer tipo de dados em formato tabular [Fisher et al. 2003]. Assim, o RIST é um *middleware* baseado no JDBC, que permite a construção de um ambiente de banco de dados confiável e seguro, a partir do uso de bases de dados heterogêneas e, sobretudo, sem a necessidade de efetuar modificações na aplicação, nem tampouco nos sistemas de banco de dados.

O protocolo de base do RIST é construído a partir da abstração de difusão com ordem total, que é usado como principal mecanismo para prover a consistência e integridade sistema. Para tanto, a concretização desta abstração se dá através do algoritmo de consenso denominado PAXOS Bizantino [Zielinski 2004], com as otimizações previstas para prover

a terminação do consenso com menos passos de comunicação [Martin and Alvisi 2005]. O RIST é constituído de dois componentes básicos: um driver genérico JDBC, que é usado como parte da aplicação cliente; e um controlador descentralizado que é o responsável por prover as funcionalidades de segurança e tolerância a faltas da arquitetura. Para a aplicação cliente, o driver JDBC específico é substituído pelo driver JDBC genérico do RIST, que oferece a mesma interface e transparência de acesso ao sistema de banco de dados. Por outro lado, o controlador do RIST é um componente replicado em $3f + 1$ servidores, dos quais até $f \leq \lfloor \frac{n-1}{3} \rfloor$ podem falhar de maneira Bizantina. Este controlador atua como uma espécie de *Proxy* entre o driver genérico JDBC e os sistemas de bases de dados, e expõe aos clientes uma visão equivalente a um ambiente centralizado e, sobretudo, prezando pela consistência de dados.

O modelo de transações admitido no contexto do RIST segue as premissas tradicionais e clássicas de transações. Assim, uma transação consiste de uma sequência de operações/comandos, seguidos por um comando COMMIT ou ABORT. Os comandos podem ler, atualizar, incluir e excluir entradas no banco de dados. No contexto de uma transação, o sistema assegura que as operações são executadas na base de dados, exatamente na mesma ordem em que elas foram enviadas pelo cliente. Além disso, nosso modelo estipula que o cliente aguarda a resposta de uma operação enviada, antes de submeter a(s) próxima(s). Conforme preconiza o modelo tradicional de transações, para a obtenção da semântica de transações **ACID** [Bernstein et al. 1987] é requerido que o sistema de banco de dados implemente mecanismos de controle de concorrência e de recuperação. O sistema RIST assegura o critério de consistência serializável, isto é, a execução concorrente de um grupo de transações toma o mesmo efeito que a execução sequencial destas transações, em alguma ordem.

4.1. Descrição do Protocolo de Replicação

Soluções para replicação de bases de dados podem assegurar diferentes critérios de consistência de dados. Para a concretização e correção deste protocolo consideramos o critério ***one-copy serializability*** (1-SR) [Bernstein et al. 1987]. É importante ressaltar que, em razão da primitiva de difusão com ordem total ser baseada no algoritmo de consenso PAXOS Bizantino, e do modelo clássico de transações não ter terminação garantida em ambientes assíncronos, o modelo de interação requerido para assegurar o progresso do sistema (*liveness*) é o de sincronismo terminal (*eventually synchronous system*) [Dwork et al. 1988]. Ademais, como premissa para o funcionamento e obtenção das garantias do protocolo, é necessário que o sistema de banco de dados implemente o controle de concorrência baseado no bloqueio de duas fases (*two-phase locking*) e critério de consistência baseado em serialização (*serializable*).

Conforme ilustrado na Figura 1, o princípio de funcionamento do protocolo de replicação da arquitetura RIST é o seguinte:

Início: o início de uma transação ocorre a partir do momento em que o cliente, por intermédio do componente de *middleware* abre uma conexão no sistema de banco de dados. Neste momento, o subsistema do cliente envia de forma transparente uma mensagem **BEGIN** por meio da primitiva de difusão com ordem total ao conjunto de réplicas e, ao entregarem esta mensagem as réplicas definem de forma determinista qual delas será a líder desta transação, que terá a tarefa de executar as operações da transação. Ao final deste processo as réplicas enviam uma mensagem de confirmação ao cliente, para atestar que a transação foi iniciada com êxito (passos 1 e 2 da Figura 1).

Execução: quando o cliente recebe a confirmação de início da transação, a mensagem de confirmação traz consigo a indicação da réplica líder da transação. A partir deste momento até o final da transação, o cliente interage apenas com a réplica líder, isto é, ele envia as operações que fazem parte da transação apenas à réplica líder, caracterizando o que é conhecido como execução especulativa [Lampson 2006] ou execução otimista. As demais réplicas passam a ter parte no contexto da transação a partir do momento em que a aplicação solicita a confirmação para a transação (passos 3 a 6 da Figura 1).

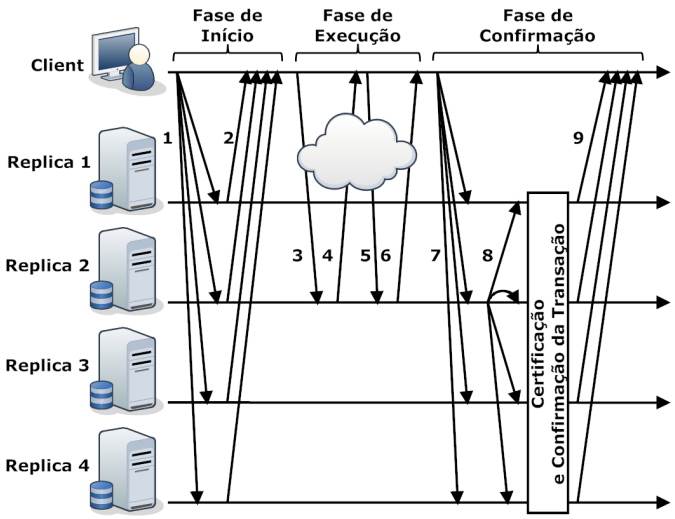


Figura 1. Funcionamento do protocolo

Confirmação: a confirmação da transação é ilustrada nos passos 7-9 da Figura 1. Ao solicitar a confirmação da transação, o componente de *middleware* da aplicação cliente envia a mensagem do pedido de confirmação por meio de difusão com ordem total, momento este onde também é realizada a propagação da transação para as demais réplicas do grupo. A mensagem do pedido de confirmação (que é assinada) traz consigo as operações executadas pela transação e o resumo criptográfico dos respectivos resultados (para fins de comparação). Assim, quando a réplica líder daquela transação recebe o pedido de confirmação, ela repassa através de difusão com ordem total o mesmo pedido de confirmação, adicionando os RS e WS produzidos pela transação, de modo a evitar que um cliente malicioso possa forjar um pedido de confirmação, ou ainda, que possa enviar pedidos de confirmação espúrios. A propagação através da difusão com ordem total assegura que todas as réplicas corretas recebam a confirmação da transação e a concretizem na mesma ordem, visando preservar o critério de serialização. A partir do recebimento da confirmação enviada pela réplica líder, as demais réplicas primeiramente verificam a consistência das mensagens enviadas (assinaturas, resumos, autenticação, etc.), e se é detectada qualquer anomalia no processo, a transação é descartada, ou do contrário, é executada. Ao iniciar a execução local da transação, as réplicas primeiramente adquirem os bloqueios de leitura e de escrita para os itens de dados afetados pela transação, e checam se RS e WS está em consonância com as operações enviadas, pois, no caso de um líder malicioso, o mesmo poderia repassar dados inválidos para as demais réplicas. Concluídas as verificações, as réplicas bloqueiam os dados e executam as operações sobre os mesmos. Após a execução, as réplicas submetem a transação a um teste de certificação, que irá verificar se não houve a violação das propriedades de serialização, isto é, se não houve conflitos da transação em questão, com outras transações concorrentes[3]. Em não havendo conflitos, as réplicas verificam se os resultados obtidos na

[3]Uma transação T_i é considerada concorrente em relação a T_j, se ela não precede T_j ($T_i \nrightarrow T_j$)

execução da transação localmente estão em consonância com aqueles enviados pela réplica líder ao cliente, cujos resumos (*hash*) dos resultados foram enviados na mensagem do pedido de confirmação, e em sendo as réplicas efetivam a transação, do contrário a transação é desfeita.

4.2. Componentes Arquiteturais

A ilustração da figura 2 apresenta os componentes do *middleware* RIST que se encontram em cada réplica do sistema. Todavia, cabe ressaltar que a arquitetura em questão é distribuída e os diversos sistemas de gerência de bases de dados (DBMS) da figura representam a capacidade do *middleware* em operar sobre diversas plataformas de SGBD. A arquitetura RIST é formada por uma série de componentes, que atuam em conjunto a fim de manter as propriedades, bem como prover as garantias propostas para o sistema. Esta seção descreve cada um dos componentes da arquitetura proposta.

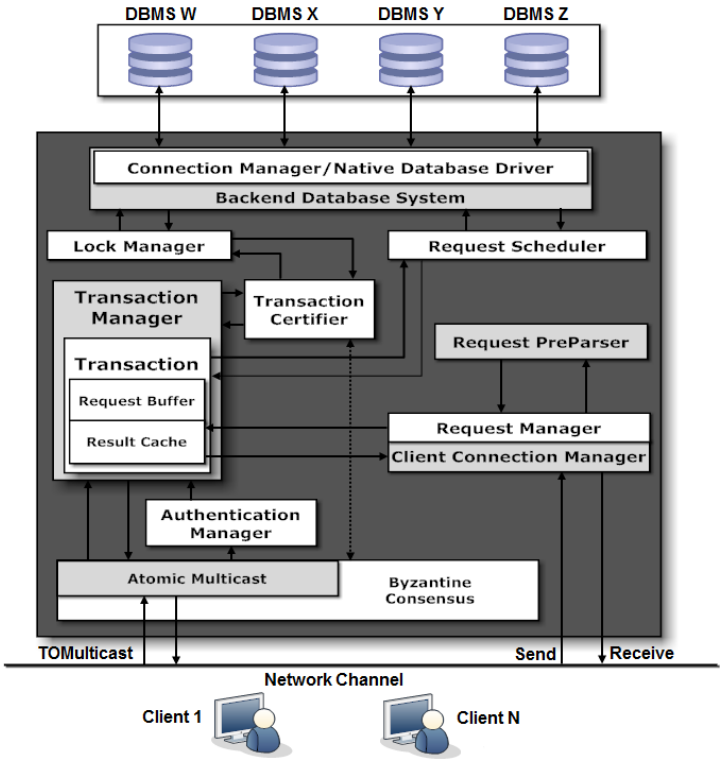


Figura 2. Componentes de uma réplica do RIST

4.2.1. Gerenciador de Conexões Cliente (*Client Connection Manager*)

O gerenciador de conexões cliente é o módulo responsável por estabelecer o canal de comunicação entre a réplica líder de uma dada transação, e o respectivo cliente emissor. Este componente opera diretamente sobre as abstrações de comunicação de médio nível, tal como *sockets* e *channels*. Assim, cabe ao módulo o recebimento das requisições/operações enviadas no contexto de uma transação e encaminhá-las ao módulo de gerenciamento de requisições para posterior tratamento e execução da operação.

4.2.2. Gerenciador de Requisições (*Request Manager*)

O módulo gerenciador de requisições é quem estabelece o vínculo da requisição ao contexto da transação a que ela pertence. A cada requisição recebida em uma transação, o módulo

primeiramente realiza a verificação acerca da sintaxe da requisição enviada para certificar se ela está bem formada, isto é, se ela segue os padrões de sintaxe definidos e aceitos para a execução pelo sistema. Uma vez que a requisição é aceita pela avaliação preliminar ela é direcionada para a transação a que pertence para posteriormente ser enviada para execução no sistema de banco de dados, pelo módulo escalonador.

4.2.3. Pré-Analisador de Requisições (*Request PreParser*)

Este componente é responsável por realizar a pré-avaliação da sintaxe do comando enviado pelo cliente, para processamento pelo sistema. Todavia, como o RIST preconiza os aspectos de interoperabilidade, as réplicas do sistema podem operar sobre sistemas de bancos de dados heterogêneos, isto é, de diferentes fornecedores. Diante disso, surgem alguns problemas que devem ser previstos e tratados pelo RIST. O primeiro deles decorre do fato dos sistemas de gerência de bancos de dados implementarem diferentes dialetos para os comandos de interação com a base de dados, no caso a linguagem SQL, apesar desta possuir uma padronização conhecida como SQL ANSI. Entretanto, estipular apenas o uso (ou o aceite) de comandos escritos segundo o padrão SQL ANSI torna-se algo restritivo. Assim, este problema é resolvido no RIST através do mapeamento do comando enviado pelo cliente para a sintaxe nativa do sistema de banco de dados da réplica em questão. E para prover melhor desempenho do processo de pré-análise de requisições, as réplicas mantém um *cache* de comandos pré-avaliados e já traduzidos para execução, para evitar o reprocessamento dos comandos usados com maior frequência. A manutenção deste *cache* é realizada de acordo com a tradicional política de LRU (*least recently used*), onde as informações são removidas do *cache* de acordo com sua obsolescência.

4.2.4. Gerenciador de Transações (*Transaction Manager*)

Quando uma mensagem de controle de transações é entregue ao *middleware* por meio do subsistema de difusão com ordem total, este encaminha a mensagem ao módulo de gerenciamento de transações que de acordo com o contexto recebido executará a ação apropriada. As mensagens de controle do protocolo de replicação do RIST são de três tipos: (*i*) mensagem de início de transação; (*ii*) mensagem de pedido de confirmação, enviado pelo cliente; (*iii*) mensagem de confirmação, enviada pela réplica líder da transação. Assim, quando o protocolo entrega alguma destas mensagens o módulo de gerenciamento de transações realiza a criação de uma transação, bem como a alteração do estado da transação para a qual a mensagem é destinada, de acordo com o tipo da mensagem. Note que, devido ao uso do protocolo de difusão com ordem total, as ações efetuadas pelo módulo gerenciador de transações ocorrem na mesma ordem em todas as réplicas do grupo. Esta é uma premissa do sistema para que o protocolo do RIST possa prover as garantias de consistência das transações perante a ocorrência de faltas Bizantinas, e assim manter o critério de serialização.

4.2.5. Certificador de Transações (*Transaction Certifier*)

O módulo de certificação é um dos componentes mais importantes do RIST. Este módulo é o responsável por realizar o teste de certificação das transações que são submetidas para confirmação. O propósito deste módulo é determinar situações de conflito entre transações concorrentes, em face a execução otimista/especulativa realizada pelo protocolo, para prover maior escalabilidade ao sistema. Deste modo, uma transação é apta a ser confirmada se, além das verificações preliminares de consistência ela passa no teste de certificação.

O teste de certificação que é baseado na certificação de [Kung and Robinson 1981] toma como subsídios as marcações efetuadas nas transações, que ocorrem quando da entrega das mensagens *REQ-COMMIT* e *COMMIT*. Mais precisamente, uma transação T_i está sujeita à certificação se é verificada a existência de alguma transação concorrente T_j, após a entrega do pedido de confirmação para T_i. Uma transação T_j é dita corrente em relação à T_i se ela não a precede ($T_j \nrightarrow T_i$). Para facilitar o entendimento da atuação do módulo de certificação, a figura 3 ilustra a execução de quatro transações concorrentes (T_1, T_2, T_3, T_4) através do protocolo.

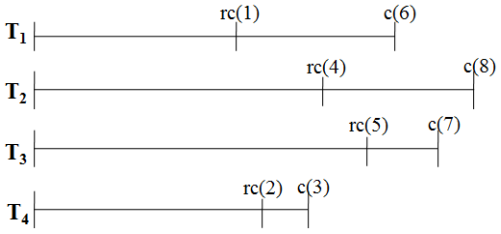


Figura 3. Execução de transações

Os eventos **rc**(n) e **c**(n) representam o momento em que ocorreu a entrega das mensagens de pedido de confirmação (*REQ-COMMIT*) e de confirmação (*COMMIT*), respectivamente. O valor n é o *timestamp* relativo à ordem de entrega das respectivas mensagens, fornecido pelo módulo de difusão com ordem total. Além da execução das transações concorrentes a figura 3 nos permite observar quais transações estão sujeitas a situações de conflitos entre si. Neste caso, em relação à transação T_4 não há transações conflitantes, pois nenhuma transação foi confirmada durante o intervalo entre o pedido de confirmação e a confirmação de fato (entre os *timestamps* 2 e 3). Por outro lado, ao analisar a transação T_1 é possível verificar que a transação T_4 é concorrente a ela, pois T_4 foi confirmada no *timestamp* 3, o que ocorreu durante o intervalo entre o pedido e a confirmação de T_1 (*timestamps* 1 e 6). Da mesma forma, T_1 é concorrente com T_3, e T_3 e T_1 são concorrentes com T_2, pelas mesmas razões.

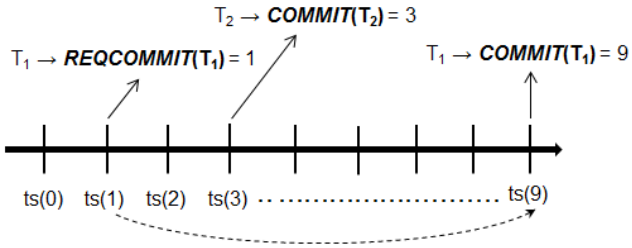


Figura 4. Relação de precedência entre transações

A figura 4 ilustra de forma mais simplificada a relação de precedência de transações, bem como quais transações devem ser consideradas pelo teste de certificação quando uma determinada transação pede confirmação. No exemplo é ilustrada a execução de duas transações identificadas por T_1 e T_2. A relação de precedência é verificada da seguinte forma: no caso de T_1, todas as transações que a precedem são aquelas que tiveram sua confirmação efetuada em algum *timestamp* anterior ao pedido de confirmação de T_1, neste caso, anterior a $ts(1)$. As transações precedentes, isto é, as que já foram confirmadas antes do pedido de confirmação para a transação em questão não são consideradas pelo teste de certificação, visto que elas já cumprem os critérios de serialização. Desta forma, apenas as transações potencialmente conflitantes, isto é, as que foram confirmadas após o pedido de confirmação da transação em questão são consideradas. No exemplo todas as transações

confirmadas a partir do *timestamp* $ts(1)$ até o *timestamp* $ts(9)$ são consideradas concorrentes com T_1 (neste caso, apenas T_2) e devem ser consideradas no teste de certificação para possibilitar a confirmação de T_1. Em suma, o teste de certificação efetuado por este módulo é que determinará se a confirmação da transação não implicará na violação do critério de serialização [Bernstein et al. 1987], de modo a preservar a integridade e consistência da base de dados. Uma transação que não passa no teste de certificação é cancelada pelo protocolo.

4.2.6. Gerenciador de Bloqueios (*Lock Manager*)

O módulo gerenciador de bloqueios é o mecanismo responsável por realizar a aquisição dos bloqueios de leitura e de escrita sobre os itens de dados afetados pela transação e que são entregues junto à mensagem de confirmação da transação. Assim, o gerenciador de bloqueios atua como uma espécie de escalonador de requisições de bloqueio sobre os itens de dados. Cabe ressaltar que como os pedidos de confirmação e a confirmação são propagados através da primitiva de difusão com ordem total, isto implica que eles ocorrem na mesma ordem em todas as réplicas corretas. E seguindo esta premissa, os pedidos de bloqueio também ocorrem na mesma ordem, de acordo com a confirmação das transações do sistema.

4.2.7. Escalonador de Requisições (*Request Scheduler*)

Quando uma nova requisição é recebida, após a execução de todo o processo de validação e associação com a respectiva transação, ela é disponibilizada ao escalonador para execução pelo sistema de banco de dados local. O escalonador é responsável por efetuar o controle de concorrência da réplica (em conjunto com o sistema de banco de dados local), bem como coordenar a execução das requisições de uma transação, de acordo com a ordem de envio das mesmas. Todas as operações executadas são síncronas com o respectivo cliente, isto é, o escalonador aguarda o recebimento da resposta da réplica líder para a requisição enviada, antes de retornar o resultado ao cliente. Neste caso, se a resposta para uma requisição não é recebida dentro de um intervalo de tempo aceitável, o escalonador retorna uma exceção como resposta à requisição. Neste caso, o cliente pode reenviar o comando para execução, se assim desejar.

4.2.8. Gerenciador de Autenticação (*Authentication Manager*)

O módulo gerenciador de autenticação estabelece o mapeamento entre o *login* e senha fornecidos pela aplicação cliente, e o *login* e senha a ser usado em cada uma das réplicas da base de dados. Assim, o *login* e senha enviados pela aplicação não são, de fato, os dados de acesso à base de dados local, mas sim uma ponte de acesso ao sistema. A funcionalidade incrementa maior segurança ao ambiente, pois os dados de acesso à base são exclusivos para cada uma das réplicas. Além disso, é permitido que cada uma das réplicas tenha diferentes logins de senhas para acesso a uma mesma base de dados.

5. Aspectos de Heterogeneidade e Não-Determinismo

Nesta seção apresentamos duas questões de fundamental importância, que decorrem do uso de réplicas heterogêneas. De acordo com o modelo de replicação de máquina de estados [Schneider 1990], é imperioso que cada comando enviado pelo cliente no contexto de uma transação seja processado de forma idêntica por cada uma das réplicas, de modo a preservar o determinismo de réplica.

Conforme mencionado na seção 4.2.3., o primeiro problema em face da heterogeneidade de réplicas de aplicação surge devido à implementação de diferentes dialetos da linguagem SQL por cada uma das réplicas de banco de dados, o que resulta em incompatibilidade, apesar desta linguagem ser padronizada pelo ANSI. Assim, é necessária a verificação preliminar e posterior tradução do comando enviado a uma determinada réplica para que o sistema possa assegurar a consistência das operações executadas em cada uma delas. Além disso, devido à algumas peculiaridades dos SGBDs tais como o tamanho dos blocos de dados, a disposição dos registros e outros, a execução dos comandos já em sintaxe correta podem retornar resultados em ordem e disposição distintas. E como a confirmação da transação requer a comparação dos resumos criptográficos dos resultados das operações, este aspecto de heterogeneidade mencionado torna-se um problema impeditivo para que a comparação dos resultados seja efetuada de forma plena nas réplicas.

Estes problemas são resolvidos no RIST através da implementação de dois mecanismos. O primeiro é a anexação de uma cláusula `ORDER BY` apropriada aos comandos SQL que já passaram pelas transformações sintáticas do módulo de pré-análise de requisições. Esta cláusula é peça fundamental para que a disposição e a ordem dos registros contidos no resultado de um comando seja a mesma para um comando em cada uma das réplicas. Além disso, para tornar os resultados das réplicas heterogêneas idênticos[4], é necessário efetuar um mapeamento sobre cada um dos resultados para uma representação uniforme. As réplicas do RIST efetuam o mapeamento dos resultados obtidos de um comando para uma cadeia de *bytes* de tamanho fixo e de representação uniforme, independente da representação específica da réplica de banco de dados. Este mapeamento ocorre após o processamento do comando e antes do envio do resultado ao cliente, para assegurar que este receba apenas resultados na representação válida do sistema.

6. Implementação, Avaliação e Resultados

Conforme mencionado na seção 4., o RIST consiste na concretização de um driver JDBC [Fisher et al. 2003], e desta forma, a implementação do protótipo do *middleware* foi realizada na linguagem Java, seguindo as especificações do JDBC. A primitiva de difusão com ordem total usada no RIST é oriunda da implementação do algoritmo PAXOS Bizantino [Zielinski 2004] que é parte do projeto *BFT-SMaRt* (*http://code.google.com/p/bft-smart*). Optou-se por usar esta implementação do PAXOS, em razão dela contemplar uma série de otimizações tais como realização do consenso em menos passos de comunicação em execuções favoráveis; acordo realizado por meio de resumos criptográficos de mensagens ao invés de fazê-lo por meio da mensagem completa; acordo por lote de mensagens ao invés de fazê-lo mensagem a mensagem.

A concretização dos canais de comunicação com os mecanismos requeridos para suportar as garantias do protocolo de replicação e execução especulativa de transações foram implementadas com o uso do *framework* Netty (*http://www.jboss.org/netty*), com vistas às otimizações e facilidades providas por este *framework*, no que compete ao desenvolvimento de aplicações baseada em passagem de mensagens. Para permitir o correto funcionamento do protocolo, bem como das comparações das respostas das réplicas durante o processo de confirmação, os resultados das operações das transações são mapeados para uma cadeia de *bytes* de tamanho fixo e posteriormente serializados para envio ao cliente. Para a serialização optamos por usar a *interface* java.io.Externalizable devido a ela apresentar maior eficiência em relação à *interface* padrão java.io.Serializable. Esta eficiência

[4]O que não ocorre naturalmente devido a representação interna de tipos de dados em cada uma das réplicas.

refletiu inclusive, em um decremento significativo no tamanho das mensagens que contém os resultados das operações.

Como é sabido que a avaliação de sistemas de computação distribuída pode ser realizada de forma analítica e/ou empírica [Jain 1991], decidimos por realizar a avaliação de nosso *middleware* nestes dois moldes. Os resultados para estas avaliações são reportados nas próximas subseções.

6.1. Avaliação Analítica

A primeira avaliação realizada em nosso *middleware* se dá em termos das complexidades algorítmicas e de algumas propriedades observadas nos trabalhos relacionados. Para esta avaliação são considerados apenas os protocolos do RIST, o HRDB [Vandiver et al. 2007], o Byzantium [Garcia et al. 2011] e o BFT-*Deferred Update* (BFT-DU) por serem os únicos a se basearem nos princípios de nossa proposta (p.ex. faltas Bizantinas). A tabela 1 descreve os resultados observados em cada um dos protocolos mencionados. Esta avaliação considerou apenas os protocolos de confirmação das soluções em razão desta ser o parte que despende o maior custo e complexidade no processamento da transação.

Tabela 1. Avaliação analítica dos protocolos de confirmação de transações

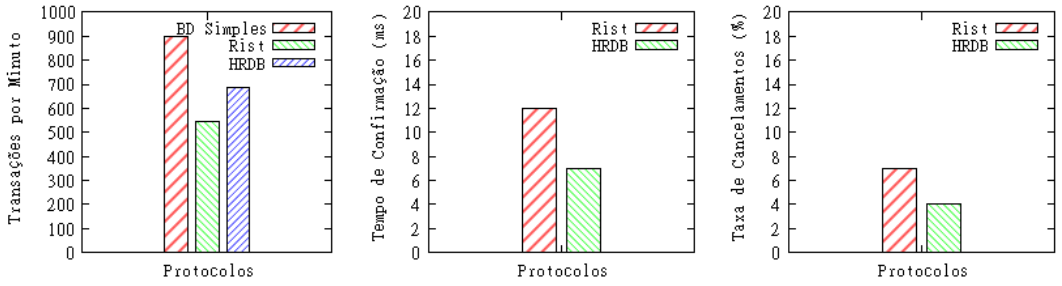
CARACTERÍSTICAS E PROPRIEDADES	PROTOCOLOS			
	HRDB	**Byzantium**	**BFT-DU**	**RIST**
Número de Réplicas	$2f + 1 + controlador$	$3f + 1$	$3f + 1$	$3f + 1$
Latência de Comunicação	4	(*TOMCast*) + 1	(*TOMCast*) + 1	2(*TOMCast*) + 1
Complexidade de Mensagens	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Consistência	Serialização (Forte)	*Snapshot* (Fraco)	Serialização (Forte)	Serialização (Forte)
Controle Centralizado	Sim	Não	Não	Não
Entidades Bizantinas	Clientes e Réplicas	Clientes e Réplicas	Apenas Réplicas	Clientes e Réplicas

A tabela 1 nos mostra que em termos de número de réplicas o HRDB é a melhor solução, já que o mesmo mantém o controle "centralizado". O HRDB não requer um acordo bizantino tal como ocorre nos demais protocolos devido a difusão com ordem total. Em termos de latência o HRDB ainda é o melhor, o que decorre do número reduzido de mensagens do protocolo também em razão do controle centralizado. O Byzantium e o BFT-DU são equivalentes e mais eficientes do que o RIST. Isso ocorre devido ao RIST usar duas difusões com ordem total (*TOMCast - Total Order MultiCast*) para o processo a confirmação, a despeito de uma difusão usada pelos demais. Contudo, o uso das duas difusões é necessário para que seja possível obter o critério de consistência serializável em face a clientes e servidores Bizantinos, uma característica não atendida em plenitude pelas demais soluções.

Em se tratando da complexidade de mensagens, todos os protocolos das soluções em que a confirmação ocorre de forma não centralizada são equivalentes e obviamente menos eficientes em relação ao HRDB. Novamente, isto decorre do uso do coordenador centralizado e da ausência do acordo entre as réplicas. O resultado para as demais soluções em termos de complexidade é oriundo do uso de protocolos de difusão com ordem total de nível inferior equivalentes (p.ex. Paxos Bizantino e PBFT). Embora os resultados gerais apontem para a perda, em termos de eficiência de nossa solução, é lícito citar que o RIST é a única solução que provê a consistência ideal (serializável) em ambientes sujeitos a faltas Bizantinas de maneira totalmente distribuída, e sem restrições quanto ao comportamento Bizantino por parte dos processos (clientes e réplicas). Deste modo, isto pode ser considerado um grande avanço em relação aos demais protocolos avaliados, o que justifica o uso do RIST a despeito da eficiência verificada.

6.2. Avaliação Experimental

A outra avaliação realizada acerca de nossa proposta e trabalhos relacionados se dá de maneira empírica, por meio da experimentação dos protocolos avaliados em um ambiente real. Todavia, esta avaliação foi realizada apenas considerando os protocolos dos *middlewares* RIST e HRDB, devido a não termos tido acesso a implementação dos demais protocolos. A avaliação foi realizada por meio do *benchmark* TPC-C, onde utilizamos uma implementação própria do *benchmark* a partir do SQL padrão, para possibilitar a execução do mesmo em qualquer sistema de gerência de base de dados que dê suporte ao SQL padrão. Para a execução do TPC-C, as bases de dados foram carregada com 10 *warehouses* e 10 *distritos* para cada uma, visando induzir maior concorrência. Os testes foram executados a partir de 20 clientes com 200 ms de intervalo entre a submissão de transações, tendo como objetivo demonstrar a aplicabilidade dos sistemas, bem como o desempenho relativo de cada um deles.



(a) Throughput. (b) Tempo de Confirmação. (c) Cancelamentos (%).

Figura 5. Avaliação experimental das soluções de *middleware*.

O ambiente no qual o *benchmark* foi realizado contou com um conjunto de 4 máquinas Itautec InfoWay ST4262, todas com a mesma composição de *hardware* e de *software*, interligadas por meio de uma rede local (LAN), e tendo como configurações: 1 microprocessador Intel®Core™2 Duo E8400 2.67GHz, 4GB RAM e uma interface Ethernet Gigabit Intel 82567LM-3. Como sistema operacional adotamos o Debian GNU/Linux 6.0, *kernel* 2.6.32-5-686 #1 SMP. Os sistemas de gerência de bases de dados usados nos experimentos foram o Informix Innovator-C Edition 11.70-UC1 e o MySQL 5.5.8, sendo duas (2) instâncias de cada um para totalizar quatro (4) réplicas (ex. $3f + 1$ réplicas para suportar 1 falta Bizantina). No experimento da figura 5(a) o valor para a **base de dados simples** é oriundo da média verificada para os resultados de uma réplica com o Informix e uma com o MySQL.

A figura 5 apresenta os resultados do TPC-C, os quais foram obtidos a partir do média de 25 execuções para cada um dos experimentos. Conforme podemos observar na figura 5(a), nosso protocolo é o que apresenta menor vazão em relação aos demais. O resultado reportado para uma única base de dados é apenas um referencial, visto que protocolos de replicação têm um custo adicional e por esta razão são menos escaláveis que bases de dados não replicadas. Em nosso caso, isso decorre do uso das duas difusões com ordem total pelo processo de confirmação do RIST. O HRDB apresenta maior vazão em virtude de ele manter o controle centralizado e não requerer o uso de difusões com ordem total e outras verificações adicionais decorrentes do acordo Bizantino (p. ex. assinaturas). Todavia, como o HRDB depende de um coordenador centralizado para conduzi-lo, isso implica na existência de um ponto de falha que não é encontrado no RIST. Além disso, como adotamos a técnica de execução otimista para as transações, o processo de confirmação do RIST despende um tempo maior em relação ao HRDB, conforme observado na figura 5(b). Isto

já era esperado devido ao modelo de propagação adotado por nosso protocolo, pois, no momento do pedido de confirmação toda a transação é propagada às demais réplicas para fins de convergência do estado das mesmas. Como no HRDB a propagação é realizada operação a operação, quando a confirmação da transação é efetuada, a réplica primária apenas compara os resultados obtidos nela e nas réplicas secundárias, para cada operação, não havendo nenhuma propagação no processo. De outro lado, há um aspecto inerente à abordagem especulativa/otimista de transações, que é o cancelamento de parcela de transações, quando em situações de concorrência se verifica a existência de conflitos de leitura e de escrita [Kung and Robinson 1981, Lampson 2006] sobre um mesmo item de dados. Cabe ressaltar que este cancelamento é necessário para que não seja violado o critério de consistência de serialização das transações que se encontram em confirmação sobre as já confirmadas. Os resultados nos mostram que o HRDB proporciona parcela menor de cancelamentos devido ao protocolo atrasar a execução de transações em que se verifica a situação de conflitos, ao contrário do que ocorre na abordagem otimista/especulativa usada no RIST.

Cabe salientar que nos resultados das figuras 5(a) e 5(b) os valores para a **base de dados simples** não foram reportados por dois motivos, o primeiro decorre do tempo de confirmação ser ínfimo em bases de dados não replicadas, e segundo, porquê o cancelamento de parcela das transações é uma implicação do uso de algumas técnicas empregadas nos protocolos de replicação, e portanto, não são verificados em bases de dados não replicadas. No entanto, apesar dos resultados apontarem para a aparente ineficiência de nosso *middleware*, cremos que há uma competitividade do mesmo em relação aos demais, principalmente devido as garantias que ele oferece. Além disso, há uma série de vantagens e desvantagens que levam a melhores ou piores resultados em termos de indicadores considerados e, portanto, cabe ao projetista do sistema decidir qual é a solução mais adequado para uma aplicação específica.

7. Conclusões

Este trabalho apresentou a proposta de um *middleware* para replicação em sistemas transacionais tolerante a faltas Bizantinas e como prova de conceito analisou de forma analítica e empírica a solução proposta e as demais existentes que se baseiam no mesmo princípio de funcionamento. Por meio destas avaliações, verificamos que a proposta é factível de uso prático, além de ser uma solução flexível para replicação de bases de dados, que visa prover escalabilidade, disponibilidade, heterogeneidade e tolerância a faltas. Além disso, devido ao uso das interfaces do JDBC, o RIST opera independente do sistema de gerência de base de dados e, sobretudo, é transparente para aplicações baseadas nesta interface.

Um aspecto importante e que deve ser relevado, é que a despeito do desempenho verificado e de algumas vantagens e custos mensurados para algumas das outras soluções em determinados aspectos, nenhuma delas contempla uma solução para replicação em sistemas transacionais completa, já que elas são limitadas em termos de consistência de dados e capacidade de tolerar faltas por parte das entidades do sistema. Diante disso, se verifica que nossa solução é a única que provê o critério de consistência ideal para sistemas de transações replicadas e, sobretudo, suporta faltas Bizantinas oriundas de clientes e réplicas a partir de um protocolo totalmente distribuído.

Referências

Bernstein, P. A., Hadzilacos, V., and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

Birman, K. and van Renesse, R. (2005). *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer.

Budhiraja, N., Marzullo, K., Schneider, F. B., and Toueg, S. (1993). The primary-backup approach. *Distributed systems (2nd Ed.)*, pages 199–216.

Cecchet, E., Julie, M., and Zwaenepoel, W. (2004). C-JDBC: Flexible Database Clustering Middleware. In *USENIX Annual Technical Conference*.

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.

Fisher, M., Ellis, J., and Bruce, J. (2003). *JDBC^{TM}API Tutorial and Reference, Third Edition*. Addison Wesley, 3 edition.

Garcia, R., Rodrigues, R., and Preguiça, N. (2011). Efficient middleware for byzantine fault-tolerant database replication. In *Proceedings of the 6th European conference on Computer Systems - EuroSys'11*. ACM.

Gray, J., Helland, P., O'Neil, P., and Shasha, D. (1996). The dangers of replication and a solution. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 173–182, New York, NY, USA. ACM.

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

Kung, H. T. and Robinson, J. T. (1981). On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6:213–226.

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

Lampson, B. W. (2006). Lazy and speculative execution in computer systems. In Shvartsman, A. A., editor, *OPODIS'06: Proceedings of the 10th International Conference on Principles of Distributed Systems*, volume 4305 of *LNCS*, pages 1–2. Springer-Verlag.

Lung, L. C. (2009). Tolerância a intrusões em sistemas de computação distribuída. In *II Seminário sobre Grandes Desafios da Computação no Brasil*, pages 13–16.

Martin, J.-P. and Alvisi, L. (2005). Fast Byzantine consensus. In *Proceedings of the Dependable Systems and Networks - DSN 2005*, pages 402–411.

Pedone, F., Guerraoui, R., and Schiper, A. (2003). The database state machine approach. *Distributed and Parallel Databases*, 14(1):71–98.

Pedone, F., Schiper, N., and Armendáriz-Iñigo, J. (2011). Byzantine fault-tolerant deferred update replication. In *Proceedings of the 5th Latin-American Symposium on Dependable Computing - LADC'11*. SBC.

Schiper, A. and Raynal, M. (1996). From group communication to transactions in distributed systems. *Communications of the ACM*, 39:84–87.

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

Skeen, D. (1981). Nonblocking commit protocols. In *SIGMOD '81: Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, pages 133–142, New York, NY, USA. ACM.

Vandiver, B., Balakrishnan, H., Liskov, B., and Madden, S. (2007). Tolerating byzantine faults in transaction processing systems using commit barrier scheduling. In *SOSP'07: Proceedings of 21st ACM Symposium on Operating Systems Principles*.

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