Adaptação Dinâmica de Protocolos de Consenso Bizantino

Carlos Carvalho, Daniel Porto, Luís Rodrigues, and Alysson Bessani

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{carlosacarvalho, danielporto, ler}@tecnico.ulisboa.pt
LASIGE, Faculdade de Ciências, Universidade de Lisboa
anbessani@ciencias.ulisboa.pt

Resumo O problema do consenso distribuído na presença de faltas Bizantinas tem recebido particular atenção nas últimas décadas. Existem hoje diversos protocolos para este efeito, cada um otimizado para condições de execução particulares. Uma vez que na maioria dos casos os sistemas reais operam em condições dinâmicas, importa desenvolver mecanismos que permitam adaptar os protocolos em tempo de execução ou substituir um protocolo por outro mais adequado às condições correntes. O problema da adaptação dinâmica de protocolos de consenso não é novo, mas a literatura é escassa para o caso Bizantino e não existem trabalhos que permitam comparar as soluções existentes. Este trabalho tem dois objetivos complementares. Em primeiro lugar, estuda como as diferentes técnicas de adaptação dinâmica propostas para o modelo de falta por paragem podem ser aplicadas na presença de faltas Bizantinas. Em segundo lugar, através da concretização destas técnicas numa moldura de software comum, baseada no pacote de código aberto BFT-SMaRt, apresenta um estudo comparativo do desempenho das mesmas.

1 Introdução

A replicação de máquinas de estado (RME)[19] é uma das técnicas fundamentais para fornecer tolerância a faltas. No seu núcleo, esta técnica pressupõe a execução de um protocolo de consenso distribuído para que as réplicas acordem na ordem pela qual devem processar os pedidos. Este trabalho foca-se no caso em se pretende usar RME para tolerar faltas Bizantinas (TFB). A solução do consenso na presença de faltas Bizantinas recebeu nas últimas décadas uma atenção considerável da comunidade científica, que resultou no desenvolvimento de um conjunto significativo de protocolos distintos, cada um otimizado para condições de execução particulares.

Neste trabalho estamos interessados no estudo de mecanismos que permitam adaptar, ou substituir, em tempo de execução um protocolo de consenso por outro mais adequado às condições de operação correntes da RME. Isto é relevante pois a maioria das aplicações práticas de RME estão sujeitas a variações do seu ambiente de execução, desde alterações na carga imposta pelos clientes a alterações no estado da rede que interliga os várias réplicas. Não existindo um

protocolo que supere todos os outros para um leque de condições de operação alargado, a única forma de assegurar um bom desempenho do sistema consiste em realizar a sua adaptação dinâmica.

O problema da adaptação dinâmica de protocolos de consenso não é novo, mas a literatura é escassa para o caso Bizantino e vários dos mecanismos propostos necessitam de ser modificados para operarem neste cenário. Mesmo entre aqueles desenvolvidos para contextos Bizantinos não existe, tanto quanto sabemos, nenhum trabalho que permita comparar o seu desempenho. Desta forma, quem procure suportar a adaptação dinâmica tolerante a faltas Bizantinas, não tem ao seu dispor dados concretos para poder escolher a técnica de adaptação que melhor se adapta às características e objetivos do sistema alvo.

Este trabalho tem, assim, dois objetivos principais. Em primeiro lugar, estuda como as diferentes técnicas de adaptação dinâmica que foram propostas para o modelo de falta por paragem podem ser aplicadas na presença de faltas Bizantinas. Em segundo lugar foram concretizadas algumas destas técnicas num sistema comum, o BFT-SMaRt[3], por forma a ser possível compará-las experimentalmente. Neste contexto, o artigo apresenta um estudo comparativo do desempenho desta técnicas

2 Trabalho Relacionado

Entre os sistemas que foram propostos para concretizar replicação de máquina de estados TFB destacamos o PBFT [6], o Aardvark [9] e o Zyzzyva [11]. Cada um destes sistemas opera melhor em certas condições, sendo pior noutras, não havendo nenhum que supere todos os outros em todas as situações, como demonstrado por Singh et. al. [20]. O Zyzzyva tem melhor desempenho quando não ocorrem faltas e a rede é estável. Por outro lado, quando ocorrem faltas frequentemente o Aardvark opera melhor que os restantes, sacrificando o desempenho no caso livre de faltas. Para além disto, o desempenho do PBFT é menos sensível ao aumento do tamanho da mensagens trocadas, quando comparado com o Zyzzyva. Estas diferenças motivam o interesse de comutar entre diferentes protocolos, ou adaptar o algoritmo em execução, tirando partido das características de cada um.

Podemos dividir os sistemas que suportam a comutação de protocolos em três grandes categorias, a saber: i) sistemas que suportam a reconfiguração de protocolos monolíticos cientes da adaptação; ii) sistemas que permitem comutar entre protocolos com suporte explicito para desativação; iii) sistemas que permitem comutar entre dois protocolos arbitrários. Nos parágrafos seguintes discutimos estas diferentes aproximações ao problema da adaptação dinâmica de protocolos de consenso.

Os sistemas que suportam a reconfiguração de protocolos monolíticos cientes da adaptação assumem a existência de um único protocolo que já incorpora topos os comportamentos que se pretendem ativar em tempo de execução. Neste caso, a adaptação dinâmica consiste simplesmente em modificar um ou mais parâmetros de configuração do protocolo. Apesar deste ser o caso mais simples,

não é completamente trivial. Uma vez que os protocolos de consenso envolvem múltiplos processos, é necessário coordenar a reconfiguração das diferentes réplicas, para assegurar que estes operam em configurações compatíveis (tipicamente, com a mesma configuração). A técnica mais simples para conseguir esta coordenação, sugerida em [13], consiste em usar o próprio protocolo de consenso para definir o instante lógico em que a reconfiguração tem efeito. Para isso, a máquina de estados replicada deve suportar comandos de reconfiguração, que não são entregues à aplicação, mas que aplicam as alterações aos parâmetros, de forma a que estas alterações sejam totalmente ordenadas em relação ao fluxo das mensagem de dados. Por exemplo, o ByTAM[18], um sistema anterior que deu os primeiros passos no caminho de produzir uma versão reconfigurável do BFT-SMaRt, usa esta técnica. Uma limitação desta aproximação é que obriga a que todos os comportamentos sejam suportados por um único protocolo, de enorme complexidade e cuja correção é difícil de assegurar.

Uma solução mais modular, consiste em concretizar os diferentes comportamentos usando protocolos independentes e depois ter mecanismos para comutar entre protocolos distintos. Designamos o módulo que permite comutar entre dois ou mais protocolos um *comutador*. A comutação pode ser facilitada se os protocolos exportarem uma interface que permita ao comutador desativar o protocolo, colocando-o num estado quiescente; na literatura, estes protocolos são designados por desativáveis (do Inglês, "*stoppable*"[12]). Neste caso, após a desativação ter sido solicitada pelo comutador, as várias réplicas do protocolo coordenam-se entre si para assegurar que novas mensagens já não serão processadas. Desta forma a comutação entre dois protocolos A e B pode ser executada da seguinte forma: antes da reconfiguração o comutador envia mensagens para o protocolo A; quando a reconfiguração se inicia, o comutador para de submeter mensagem e solicita a desativação do protocolo A: o comutador interrompe então o fluxo de mensagens, enquanto espera pela garantia que o protocolo A já está num estado quiescente; depois de obter a confirmação de desativação do protocolo A, o comutador retoma o envio de mensagem, agora através do protocolo B. Note-se que pode existir um hiato na comunicação durante a reconfiguração, uma vez que desativação de um protocolo não é instantânea e exige coordenação entre as diversas réplicas. Para além disso, nem todas as concretizações disponíveis de protocolos de consenso Bizantino possuem suporte para a sua desativação, o que limita a cobertura desta aproximação.

Aublin et. al. apresentaram uma extensão a este conceito, propondo algoritmos com suporte explicito para desativação[1]. Este tipo de algoritmos permite parar a sua execução quando uma determinada condição acontece, não estando limitados a parar devido à receção de um comando. Usando estes algoritmos, os autores propõem o Abstract, uma moldura de desenvolvimento de sistemas TFB que permite desativar um algoritmo e substituí-lo por outro quando as condições de ambiente mudam. Quando um algoritmo é desativado, passa a responder a todos os pedidos com uma prova do seu término, o seu histórico de operações e uma indicação do próximo algoritmo a ser ativado. O cliente é então responsável por reencaminhar o seu pedido, em conjunto com os restantes dados para

o novo algoritmo a ser usado. Esta abordagem contínua a trazer um hiato na comunicação, devido à necessidade de retransmissão de pedidos. Não usando um *comutador* dedicado, necessita também de transferir mais dados pela rede (o histórico), a fim de garantir a correção do sistema como um todo.

Finalmente, é possível comutar entre protocolos distintos que são modelados como "caixas pretas" e que não oferecem nenhum suporte para a adaptação. A dificuldade deste cenário é que não existe uma maneira direta de um comutador identificar se um dado protocolo já se encontra quiescente, pelo que os comutadores nas diferentes réplicas têm que se coordenar explicitamente. Uma maneira de fazer esta coordenação consiste em usar um dos protocolos para enviar mensagens de controlo (designadas por *marcadores*) que são desta forma ordenadas de forma total em relação ao fluxo das mensagens de dados. Em particular, para iniciar a comutação de A para B, cada comutador pode enviar um marcador através do protocolo A; quando o primeiro marcador for entregue, todos os comutadores passam a entregar apenas as mensagens recebidas através do protocolo B. Infelizmente, devido à assincronia do sistema, um comutador pode ver várias das mensagens enviadas por si para o protocolo A ordenadas depois dos marcadores, sendo neste caso obrigado a re-enviar essas mensagens para o protocolo B. Desta forma, pode existir não só um hiato durante a reconfiguração, como um também aumento no uso da rede devido à necessidade de re-enviar mensagens de dados.

Uma estratégia para mitigar o hiato na comunicação que a reconfiguração pode introduzir consiste em, durante a reconfiguração, enviar todas as mensagens em paralelo para o protocolo A e B. O método para definir o momento lógico em que se dá a comutação pode continuar a ser o descrito anteriormente. No entanto, quando a comutação é feita, as mensagens não ordenadas por A já estarão ordenadas por B e prontas a entregar. Uma variante desta estratégia foi sugerida por [16] para protocolos tolerantes a faltas por paragem em sistemas com um detetor de faltas perfeito. Uma variante da mesma para sistemas assíncronos é discutida em [4]. Como é óbvio, uma desvantagem desta estratégia é que acarreta um aumento significativo da utilização da largura de banda durante a reconfiguração uma vez que todas as mensagens são ordenadas pelos dois protocolos, pelo que só pode ser aplicada em sistemas onde a rede não constitua um ponto de estrangulamento.

É de notar que uma técnica que utilize protocolos cientes da adaptação permite, com maior facilidade, introduzir ajustes mais finos ao modo de execução do sistema. Isto é, por exemplo, se se desejar trocar o líder do algoritmo, é possível fazê-lo alterando apenas um parâmetro da configuração. Por outro lado, em soluções recorrendo a protocolos das categorias desativáveis e caixa preta é necessário criar uma nova instância do protocolo com uma configuração inicial diferente. Para além disto, em muitos casos estes parâmetros nem são configuráveis de todo, não sendo alteráveis sem modificar o código dos algoritmos.

3 Adaptação de Protocolos em Contexto Bizantino

Nesta secção discutimos a concretização de diversos comutadores de protocolos na presença de faltas Bizantinas. Quando os algoritmos de comutação não foram originalmente propostos para este modelo, apresentamos quais as alterações que foram necessárias aplicar aos mesmos. Apresentamos também algumas novas otimizações não discutidas na literatura.

3.1 Modelo do Sistema

Assumimos o modelo de faltas Bizantinas, em que os processos que falham podem apresentar um comportamento arbitrário, incluindo entrar em conluio para atacar o sistema de forma coordenada. Não obstante, consideramos um adversário com recursos computacionais limitados, e que não consegue quebrar as primitivas criptográficas usadas pelos protocolos. Por fim, assume-se uma rede parcialmente síncrona, em que podem existir períodos de assincronia arbitrários, mas que alguma vez existe um período de sincronia em que o sistema pode fazer progresso; nos momentos de sincronia, as mensagens transmitidas são entregues ao destinatário dentro de um intervalo de tempo conhecido.

Assumimos que no sistema existem N réplicas com a capacidade de instanciar diversos protocolos de acordo Bizantino. Os clientes do serviço replicado não interagem com estas instâncias diretamente, mas sim através de um componente designado por *comutador*, responsável por encaminhar estes pedidos para um ou mais destes protocolos (tornando a adaptação dinâmica transparente para os clientes). Assumimos também que existe um componente externo ao sistema, denominado o *gestor de adaptação*, que decide qual o protocolo que vai ser usado em cada momento. Quando é necessário comutar entre protocolos, o gestor de adaptação envia uma ordem a todos os comutadores, de forma a iniciar o processo de adaptação. A concretização do gestor de adaptação é ortogonal a este trabalho, estando descrita em[17]; tipicamente o próprio gestor é também replicado e cada comutador só inicia a comutação quando receber uma ordem idêntica de um quórum destas réplicas.

3.2 Adaptação Usando Algoritmos Reconfiguráveis

A adaptação usando algoritmos reconfiguráveis, apresentada por Lamport et. al.[13], é genérica e pode ser aplicada num contexto Bizantino sem grandes alterações. A correção da comutação depende exclusivamente das propriedades do protocolo de consenso subjacente, pelo que reconfigurarmos um protocolo tolerante a faltas Bizantinas, a própria reconfiguração também tolera este tipo de faltas. Como foi referido anteriormente, esta técnica é potencialmente a mais eficiente, tanto mais que o protocolo pode ser concretizado de forma a dar prioridade aos pedidos de reconfiguração, fazendo com que estes pedidos sejam ordenados antes de outras mensagens que estejam em fila de espera para ser ordenadas.

3.3 Adaptação Usando Algoritmos Desativáveis

Num sistema tolerante a faltas por paragem, uma maneira simples de efetuar a troca entre dois protocolos desativáveis, A e B respetivamente, consiste em executar o seguinte algoritmo: cada comutador solicita a desativação do protocolo A e inibe-se de enviar mais mensagem até receber a confirmação de que A está quiescente, momento em que reinicia o envio de mensagem pelo protocolo B. No caso Bizantino, este algoritmo apresenta o seguinte problema, que advém da maioria dos protocolos de consenso Bizantino se basearem na eleição de um processo líder: considere-se o caso em o processo p_i é o líder do protocolo B mas é o último a ser informado que A se encontra no estado quiescente. Os restantes processos, tendo já comutado para o protocolo B podem, erradamente, assumir que ausência de atividade do líder se deve a uma falta deste (quando, de facto, p_i está correto mas bloqueado à espera da desativação de A). Isto pode dar início a processos de mudança de líder no protocolo B mesmo antes da sua operação ser iniciada em pleno. Por esta razão, num sistema tolerante a faltas Bizantinas, pode ser preferível que um comutador passe de imediato a enviar mensagens no protocolo B, mesmo antes de ter a certeza que o protocolo A está quiescente. Note-se que as mensagens entregues por B terão que ser colocadas em quarentena até que A fique quiescente, para evitar entregar à aplicação a mesma mensagem mais que uma vez (as mensagens que entretanto forem ordenadas por A, devem ser descartadas da lista de mensagens ordenadas por B).

3.4 Adaptação Usando Algoritmos Não Adaptáveis

Quando é necessário comutar entre algoritmo do tipo "caixa-preta", que não fornecem nenhum suporte para a adaptação, é necessário enviar um marcador (uma mensagem de controlo dedicada ou uma *flag* associada a uma mensagem comum) no protocolo A para decidir qual o instante lógico, no fluxo de mensagens, em que se comuta para o protocolo B. Uma dificuldade adicional na presença de faltas Bizantinas, consiste em verificar que o marcador corresponde a uma reconfiguração que foi de facto solicitada pelo gestor de adaptação, para evitar que um comutador Bizantino possa induzir comutações indesejáveis. Considerámos duas técnicas para obter esta garantia. A primeira consiste em incluir no marcador uma prova de veracidade do pedido de configuração; esta prova consiste do comando de reconfiguração assinado por um quórum de gestores de adaptação. Outra alternativa seria iniciar a comutação apenas após receber $f + 1$ marcadores de diferentes comutadores. Esta última opção dispensaria a necessidade de enviar a prova (uma vez que ao esperar por $f + 1$ marcadores, temos a certeza que um deles foi enviado por um processo correto), mas atrasaria o processo de comutação. Por esta razão, no nosso protótipo usamos a primeira solução.

3.5 Comutação com Paralelização

Vimos anteriormente que uma maneira de reduzir o hiato que pode ocorrer durante a comutação consiste em enviar todas as mensagens nos dois protocolos

durante o processo de reconfiguração. Os trabalhos que recorrem a esta técnica, como [16] e [4], assumem que quando o processo de comutação se inicia os protocolos A e B já estão instanciados em todas as réplicas e que nenhum processo usa estes protocolos desnecessariamente. No contexto Bizantino, existe o risco de um processo Bizantino começar a transmitir no protocolo B mesmo que a comutação não tenha sido solicitada pelo gestor de adaptação, o que levará a um desperdício de recursos que pode ser ampliado se os processos remotos passarem a os processar e responder, viabilizando um ataque de negação de serviço. Para minimizar este problema, desenvolvemos o seguinte estratégia. Quando um processo envia uma primeira mensagem no protocolo B deve agregar a essa mensagem uma prova de que a comutação para B foi solicitada. Tal como anteriormente, esta prova consiste do comando de reconfiguração assinado por um quórum de gestores de adaptação. Um processo que recebe pela primeira vez uma mensagem para o protocolo B ativa o mesmo se a prova for válida ou descarta a mensagem e emite uma acusação ao seu emissor, se a prova for inválida.

4 Concretização

De forma a realizar uma avaliação experimental, que suporte uma análise comparativa do desempenho das diversas técnicas referidas anteriormente, concretizamos as mesmas numa moldura de software comum, o BFT-SMaRt[3]. O BFT-SMaRt é um pacote de software aberto que permite executar um serviço de máquina de estados replicada tolerantes a faltas Bizantinas. Escolhemos este pacote como moldura para concretizar os comutadores uma vez que é uma das poucas concretizações de RME TFB que mantém uma equipa de suporte, e também uma das concretizações com melhor desempenho. Infelizmente, o BFT-SMaRt oferece um único protocolo de consenso, o Mod-SMaRt [21]. Desta forma, para ilustrar a comutação entre protocolos distintos, desenvolvemos também um protocolo adicional para o BFT-SMaRt, em particular desenvolvemos uma concretização parcial do "Fast Byzantine Consensus" descrito em [15], que denominamos Fast-SMaRT. Para além disso, acrescentamos ao BFT-SMaRt um módulo comutador, que medeia a interação entre os clientes e os protocolos de consenso suportados. O comutador está preparado para receber comandos de reconfiguração de um gestor de adaptação replicado, cujo desenvolvimento está a ser feito por outros elementos da nossa equipa[17].

Para além do comutador e do novo protocolo de consenso já referido acima, desenvolvemos também outras extensões ao BFT-SMaRt importantes para as nossas experiências. Nomeadamente, desenvolvemos versões desativáveis dos protocolos Mod-SMaRt e Fast-SMaRt para podermos testar a comutação entre protocolos com suporte para desativação, assim como a possibilidade de reconfigurar o líder destes protocolos em tempo de execução que permite comparar o tempo entre reconfigurar um único protocolo ou comutar entre duas instâncias do mesmo protocolo com diferentes configurações. Foi também necessário prover o BFT-SMaRt com suporte para a execução paralela de múltiplos protocolos; isto foi conseguido acrescentando novos campos aos cabeçalhos das mensagens e

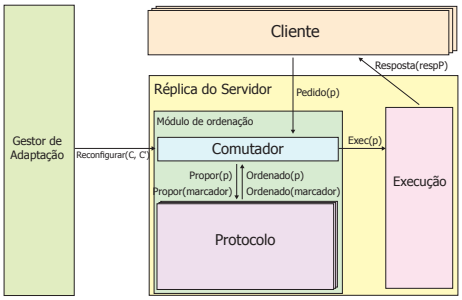


Figura 1: Arquitetura do sistema de adaptação desenvolvido.

uma camada de despacho que encaminha as mensagens para as instâncias certas. Finalmente, foi concretizado suporte para incluir nas mensagens uma prova de que a reconfiguração foi solicitada, para permitir a ativação por necessidade de novas instâncias em processo que, devido à assincronia do sistema, ainda não receberam estes comandos diretamente das réplicas do gestor de adaptação.

Note-se que seria possível otimizar a comutação tendo comutadores específicos para diferentes tipos de faltas (por exemplo, concretizando os protocolos descritos em [16] para comutar entre duas configurações tolerantes a faltas por paragem). No entanto, o protocolo atual tolera quer faltas Bizantinas quer faltas por paragem, podendo ser usado para comutar entre as várias configurações possíveis.

Ao desenvolver estas extensão ao BFT-SMaRt, tivemos a preocupação de as integrar da melhor forma no atual tronco de desenvolvimento. Por exemplo, o BFT-SMaRt estava já preparado para receber alguns comandos de reconfiguração, se bem que apenas para algumas adaptações específicas como, por exemplo, a alteração em tempo de execução do número de réplicas. Desenvolvemos os nossos mecanismos de suporte à adaptação, em particular o comutador, como uma extensão e generalização destes serviços. A concretização do novo protocolo seguiu também a decomposição em classes já usada na concretização do Mod-SMaRt. Uma representação da arquitetura geral do sistema é apresentada na Figura 1. Estas opções irão facilitar a manutenção do nosso código e a sua inclusão futura na distribuição do BFT-SMaRt. O código desenvolvido corresponde a aproximadamente 2K novas linhas de código (a distribuição base do BFT-SMaRt tem cerca de 28K linhas). Uma parte significativa do esforço de desenvolvimento das nossas extensões consistiu na necessidade de compreender em profundidade o código fonte do BFT-SMaRt para assegurar a integração correta das novas extensões.

5 Avaliação

Por forma a comparar as diferentes técnicas de adaptação, nesta secção pretendemos responder às seguintes questões: *a)* qual é a diferença no tempo que leva

uma reconfiguração a ser executada usando as diferentes técnicas? *b)* qual é o impacto no desempenho do sistema provocado pela reconfiguração? *c)* que carga é introduzida na rede durante o processo de reconfiguração?

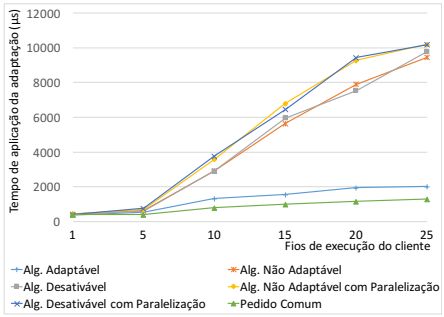
Para executar as experiências foi utilizado o *micro-benchmark* do BFT-SMaRt com 6 réplicas, o número mínimo exigido pelo Fast-SMaRt para tolerar uma falta bizantina, e um cliente capaz de introduzir carga variável no sistema. Todas as réplicas e o cliente foram hospedados em máquinas virtuais independentes no serviço DigitalOcean. Cada máquina têm dois núcleos de processamento a 2.40GHz, 2GB de memória RAM e uma ligação de rede de 1Gb/s *full-duplex*. Esta configuração foi escolhida por ser a mais poderosa, logo próxima de um servidor real, dentro dos recursos disponíveis. O gerador de carga do *micro-benchmark* envia pedidos em diferentes fios de execução, simulando múltiplos clientes. Em todas as experiências cada cliente enviou pedidos de 1KB sucessivos a cada resposta (de 10B), sem intervalo de tempo entre si (*close-loop*). O sistema foi deixado em execução durante 4 minutos antes de ser efetuada qualquer adaptação, por forma a permitir que a carga introduzida pelo cliente fizesse efeito e o sistema atingisse o seu desempenho em regime estável. Foram também feitas as experiências com 11 réplicas, tolerando duas faltas e, embora o débito do sistema tenha sido 62% inferior, o padrão observado é semelhante; estes resultados são aqui omitidos devido à falta de espaço, estando disponíveis em [5].

5.1 Tempo de Reconfiguração

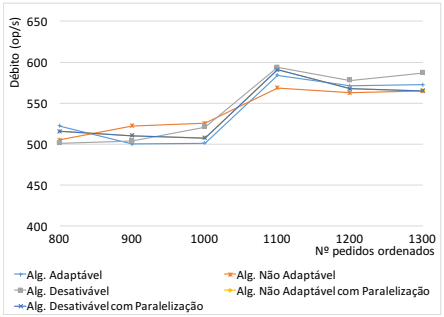
Para avaliar o tempo de reconfiguração de cada técnica foi contabilizado o tempo desde que chega um pedido de reconfiguração ao sistema até que esta é efetivamente aplicada, isto em função da carga no sistema. Os resultados são apresentados na Figura 2a. No gráfico encontra-se também representado o tempo de ordenação de um pedido comum, desde que é retirado da fila de espera até que é terminada a sua ordenação. Pode-se observar que o tempo de adaptação usando algoritmos adaptáveis cresce muito mais lentamente do que as restantes soluções. Mais tarde discutimos as causas para este comportamento.

5.2 Influência da Adaptação na Carga na Rede e no Débito

Para avaliar o impacto de executar uma adaptação com cada uma das soluções desenvolvidas foi usado um cliente com 20 fios de execução, tendo sido a carga que verificámos ser visível no sistema sem o colocar num estado de sobrecarga. Foram aferidas duas métricas: o débito de operações ordenadas por segundo e a carga na rede. O débito foi calculado a cada 100 pedidos ordenados, tendo por base o tempo decorrido desde o fim da ordenação dos 100 pedidos anteriores. Após o pedido 1000 foi submetido um pedido de reconfiguração no sistema. Os resultados obtidos estão apresentados na Figura 2b. A carga na rede foi medida durante a mesma experiência, em kB/s, com amostras a cada milissegundo. Por forma a facilitar a comparação entre as diferentes técnicas, foram recolhidos 40ms de amostras e estes foram alinhados temporalmente, sendo que o pedido

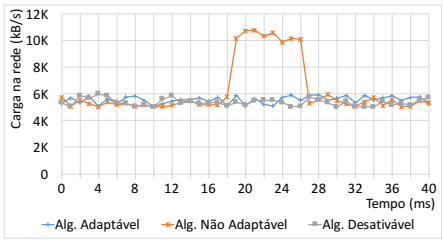


(a) Tempo de execução da adaptação

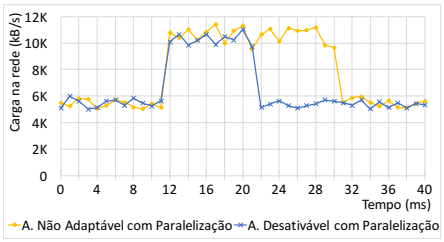


(b) Débito durante a adaptação

Figura 2: Tempo de execução da adaptação e efeito no débito



(a) Técnicas sem paralelização



(b) Técnicas com paralelização

Figura 3: Carga na rede durante um processo de adaptação

de reconfiguração chega ao sistema aos 10ms. Os dados recolhidos estão representados na Figura 3. Observa-se que, no contexto em que foram executados as experiências, nenhuma das técnicas utilizada induz uma penalização visível no débito dos sistema. Denota-se uma subida no débito entre o pedido 1000 e 1100 pois a adaptação introduzida contribuiu para um aumento do desempenho do sistema. Por outro lado, a carga imposta na rede durante a adaptação varia consoante a técnica utilizada.

5.3 Análise

Nas condições em que foram executadas as experiências, num centro de dados com uma ligação de rede de baixa latência, verifica-se que qualquer uma das técnicas de adaptação consegue adaptar o sistema sem ter um impacto negativo no seu desempenho. Contudo observam-se diferenças no tempo de aplicação de uma adaptação, bem como na carga na rede introduzida pela mesma. Uma adaptação é aplicada muito mais rapidamente se recorrermos a um algoritmo adaptável, pois é possível priorizar o comando de reconfiguração em relação aos comandos de clientes. Assim, neste caso o tempo de reconfiguração acompanha o tempo que o sistema demora a executar a ordenação de qualquer pedido. Pelo

contrário, nos restantes casos, o tempo de reconfiguração depende também do tamanho da fila de espera e da quantidade de pedidos que é necessário reenviar no novo protocolo. Esta diferença pode ser importante no desenho de um sistema se as reconfigurações tiverem por objetivo retirar o sistema de um estado de contenção ou se estas forem temporalmente sensíveis.

O uso de algoritmos não adaptáveis impõe uma sobrecarga na rede após a execução da adaptação, pois o algoritmo que deixou de estar ativo continua a executar até esgotar a fila de pedidos pendentes. Por outro lado, o uso de paralelização introduz uma carga adicional na rede antes da reconfiguração, devido à execução de ambos os algoritmos em simultâneo. Este pode ser um fator importante se o sistema operar perto do limite da largura de banda disponível, pois esta pode ser um ponto de estrangulamento do desempenho do sistema.

6 Conclusões e Trabalho Futuro

Neste artigo apresentámos e comparámos diversas técnicas de adaptação para sistemas tolerantes a faltas Bizantinas. A avaliação experimental, feita num centro de dados com redes de elevado débito, mostra que neste contexto nenhuma delas penaliza significativamente o desempenho do sistema durante a adaptação. É no entanto possível observar diferenças no tempo de adaptação e na carga da rede, pelo que a escolha da técnica a utilizar pode ser relevante em sistemas que operam no limite da utilização de recursos. Apesar destas diferenças, atendendo a que a comutação para algoritmos não adaptáveis não requer qualquer alteração aos algoritmos base, no contexto da utilização em centros de dados não parece justificar-se o esforço de desenvolvimento necessário para tornar os algoritmos adaptáveis. No entanto, este comportamento poderá ser distinto em redes de maior latência. No futuro, pretendemos realizar experiências em sistemas geo-replicados e com elevada latência na rede, para comparar o desempenho dos diferentes computadores nesse contexto. O código estará disponível em github.com/cedac/calumma.

Agradecimentos Agradecemos ao Manuel Bravo e aos revisores pelos comentários recebidos durante a preparação deste artigo. Este trabalho for parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) através dos projetos com referências PTDC/ EEI-SCR/ 1741/ 2014 (Abyss) e UID/ CEC/ 50021/ 2013.

Referências

1. Aublin, P.L., Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 bft protocols. ACM Transactions on Computer Systems (TOCS) (Jan 2015)
2. Bessani, A., Santos, M., Felix, J., Neves, N., Correia, M.: On the efficiency of durable state machine replication. In: Proceedings of the USENIX Annual Technical Conference (ATC). pp. 169–180. USENIX, San Jose, CA, USA (2013)
3. Bessani, A., Sousa, J., Alchieri, E.E.: State machine replication for the masses with BFT-SMaRt. In: Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE (Jun 2014)

4. Bortnikov, V., Chockler, G., Perelman, D., Roytman, A., Shachor, S., Shnayderman, I.: Reconfigurable state machine replication from non-reconfigurable building blocks. arXiv preprint arXiv:1512.08943 (2015)
5. Carvalho, C.: Dynamic Adaptation of Byzantine Fault Tolerant Protocols. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa (Oct 2017)
6. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI). pp. 173–186 (Feb 1999)
7. Chen, W.K., Hiltunen, M., Schlichting, R.: Constructing adaptive software in distributed systems. In: Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS). pp. 635–643. IEEE (Apr 2001)
8. Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., Riche, T.: Upright cluster services. In: Proceedings of the 22nd Symposium on Operating Systems Principles (SOPS). pp. 277–290. ACM (Oct 2009)
9. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making byzantine fault tolerant systems tolerate byzantine faults. In: Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI). pp. 153–168. USENIX Association, Berkeley, CA, USA (Apr 2009)
10. Couceiro, M., Ruivo, P., Romano, P., Rodrigues, L.: Chasing the optimum in replicated in-memory transactional platforms via protocol adaptation. IEEE Trans. Parallel Distrib. Syst. 26(11), 2942–2955 (Nov 2015)
11. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative byzantine fault tolerance. In: Proceedings of 21st Symposium on Operating Systems Principles (SOSP). pp. 45–58. ACM (Oct 2007)
12. Lamport, L., Malkhi, D., Zhou, L.: Stoppable paxos. Tech. rep., Microsoft Research (2008)
13. Lamport, L., Malkhi, D., Zhou, L.: Reconfiguring a state machine. ACM SIGACT News 41(1), 63–73 (Mar 2010)
14. Lorünser, T., Happe, A., Slamanig, D.: Archistar - a framework for secure distributed storage. http://ARCHISTAR.at (2014), gNU General Public License
15. Martin, J.P., Alvisi, L.: Fast byzantine consensus. IEEE Trans. Dependable Secur. Comput. 3(3), 202–215 (Jul 2006)
16. Mocito, J., Rodrigues, L.: Run-time switching between total order algorithms. In: Proceedings of the Euro-Par 2006. pp. 582–591. LNCS, Springer-Verlag, Dresden, Germany (Aug 2006)
17. Pasadinhas, M., Porto, D., Lopes, A., Rodrigues, L.: Adaptação guiada por políticas de sistemas tolerantes a faltas bizatinas. In: Actas do 9º Simpósio de Informática (Inforum). Aveiro, Portugal (Oct 2017)
18. Sabino, F., Porto, D., Rodrigues, L.: Bytam: um gestor de adaptação tolerante a falhas bizantinas. In: Actas do oitavo Simpósio de Informática (Inforum). Lisboa, Portugal (Sep 2016)
19. Schneider, F.: Replication management using the state-machine approach. In: Mullender, S. (ed.) Distributed Systems, 2nd Edition, chap. 7. ACM-Press, Addison-Wesley (1993)
20. Singh, A., Das, T., Maniatis, P., Druschel, P., Roscoe, T.: Bft protocols under fire. In: Proceedings of the 5th Symposium on Networked Systems Design and Implementation (NSDI). vol. 8, pp. 189–204. USENIX Association (2008)
21. Sousa, J., Bessani, A.: From byzantine consensus to bft state machine replication: A latency-optimal transformation. In: Ninth European Dependable Computing Conference (EDCC), 2012. pp. 37–48. IEEE (May 2012)