ByTAM: um Gestor de Adaptação Tolerante a Faltas Bizantinas

Frederico Sabino, Daniel Porto, and Luís Rodrigues

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{frederico.sabino,danielporto,ler}@tecnico.ulisboa.pt

Resumo Embora existam já algumas propostas para adaptação dinâmica de protocolos tolerantes a faltas bizantinas (BFT), as soluções atuais tem limitações importantes como pouca robustez, dependendo da existência de componentes centralizados e confiáveis; pouca flexibilidade, restringindo a estratégia de adaptação; e falta de extensibilidade, impossibilitando a adição de novas políticas com o serviço em operação. Para superar estas lacunas, este artigo propõe um gestor de adaptação genérico, que pode ser usado para controlar a execução de diversas políticas de adaptação, permitindo ajuste fino dos parâmetros dos protocolos. Este gestor funciona de forma independente do serviço que se pretende adaptar, sendo ele próprio tolerante a falhas bizantinas, e foi desenvolvido recorrendo a uma conhecida biblioteca de código aberto (BFT-SMaRt[1]). A sua avaliação experimental mostra que este é eficaz na adaptação, promovendo um desempenho mais eficiente do serviço BFT em diversas condições de operação.

1 Introdução

A replicação de máquina de estados com protocolos tolerantes a falhas bizantinas (do inglês, Byzantine Fault Tolerance ou BFT) é uma técnica que permite construir serviços robustos, capazes de funcionar corretamente, mesmo na ocorrência de uma minoria de falhas arbitrárias (acidentais ou maliciosas), causadas por ataques externos ou intrusões. Em essência, estes protocolos implementam uma solução do *consenso*, que é um problema fundamental em sistemas distribuídos e que tem sido extensivamente estudado na literatura [4,7,10].

Além da correção, o desempenho do protocolo tem também impacto na experiência do utilizador do serviço e, por isto, os protocolos BFT são otimizados para atingir o seu melhor desempenho nas condições operacionais mais comuns (ex. redes locais ou WAN, ocorrência esporádica de falhas, etc.); como efeito, os protocolos conhecidos são bastante penalizados quando operam fora destas condições. Por exemplo, o Zyzzyva[9] opera com melhor desempenho que o PBFT [5] quando as falhas são pouco frequentes. No entanto, a estratégia complexa do Zyzzyva para recuperação em caso de falhas acaba por torna-lo menos eficiente que o PBFT em cenários onde as falhas são mais frequentes[11].

Para além disto, os fatores que afetam o desempenho do protocolo, como o número de réplicas, a carga imposta pelos clientes, ou o nível de ameaça, apresentam valores

[1] https://github.com/bft-smart/library

que podem mudar ao longo do tempo. Desta forma, torna-se relevante o uso de técnicas que permitam a adaptação dinâmica dos parâmetros e dos algoritmos usados pelo serviços BFT. Alguns protocolos, como Aliph[8] e ADAPT[1] tentam juntar o melhor de dois mundos, permitindo mudar de um protocolo para outro em caso de falha ou alguma condição predefinida, combinando as estratégias dos protocolos mais adequadas. No entanto, são pouco flexíveis nos mecanismos de adaptação que suportam, seja na definição das políticas que disparam a adaptação, na estratégia de adaptação (mudança de protocolo, cancelamento de uma execução) e robustez (dependendo de componentes centralizados ou não-BFT). Apesar do mérito destes trabalhos pioneiros, ainda é necessário fazer bastante progresso até atingir soluções práticas, robustas e eficazes.

Neste trabalho, apresentamos o ByTAM, um gestor de adaptação genérico que permite adaptação dinâmica de protocolos. O ByTAM permite concretizar uma variedade de políticas que podem ser seleccionadas dinamicamente, podendo executar qualquer política de adaptação que possa ser expressa usando regras do tipo "condição-evento-ação". ByTAM é tolerante a falhas bizantinas, onde o gestor e a infraestrutura de gestores de adaptação operam independentemente do serviço replicado a ser adaptado. Esta arquitetura, que permite o ajuste fino dos parâmetros do protocolo em operação, exige poucas modificações dos protocolos já utilizados pelo serviço. Esta característica facilita a transição incremental de sistema legados (não adaptáveis) para novos sistemas com capacidade de adaptação dinâmica. Para além disto, o ByTAM é capaz de lidar com falhas e com a assincronia dos gestores de adaptação, factores que poderiam impedir a tomada de decisões de adaptação coerentes. O ByTAM é um projeto de código aberto, integrado com o BFT-SMaRt[3], uma biblioteca amplamente utilizada para desenvolvimento de sistemas BFT.

Nas secções seguintes deste artigo apresentaremos o ByTAM da seguinte forma: na Secção 2 faremos uma descrição mais detalhada sobre trabalhos relevantes para este artigo. Na Secção 3 apresentaremos uma visão geral do ByTAM, destacando os componentes da arquitetura e as suas principais funções. A avaliação experimental é reportada na Secção 4. Finalmente, a Secção 5 contém as conclusões deste trabalho.

2 Trabalho Relacionado

Em *Guerraoui et al.*[8], os autores apresentam uma nova abstração para reduzir o custo de desenvolver protocolos BFT denominada *Abstract* e, introduzem uma propriedade denominada *Abortability*, que permite compor diferentes instâncias de protocolos tolerantes a falhas bizantinas e alternar corretamente entre estes protocolos. Em concreto, os autores focam-se nos mecanismos necessários para substituir concretizações de protocolos BFT. Utilizando as técnicas propostas, os autores construiram um sistema denominado *Aliph* que combina três protocolos: *Quorum*[8], *Chain*[8] e PBFT[5]. Uma vez que o ênfase do trabalho não estava nas políticas de adaptação, o *Aliph* usa critérios simples para iniciar uma reconfiguração, por exemplo, quando ocorre uma falha, critérios estes que estão plasmados de forma estática no código. Desta forma, o *Aliph* inicialmente executa o *Quorum* que é mais otimista em cenários favoráveis e, quando uma falha ocorre, comuta para o *Chain* que é mais robusto porém, mais complexo. Quando outra falha qualquer ocorre, uma nova adaptação é feita, desta vez para o PBFT. Assim,

as reconfigurações são sempre executadas de acordo com uma sequência previamente definida, nomeadamente: *Quorum* $\rightarrow$ *Chain* $\rightarrow$ *PBFT*. Finalmente, o *Aliph* tenta, após um período de quarentena, retornar a executar o *Quorum*.

O ADAPT[1] é um protocolo que também explora o conceito de *Abortability*, embora de uma forma mais flexível, introduzindo uma etapa de avaliação para eleger o próximo protocolo BFT a ser ativado. O ADAPT reage às mudanças das condições do ambiente através da introdução de técnicas de Aprendizagem Automática para guiar o processo de seleção. Desta forma, uma adaptação pode ser ativada mediante métricas designadas por *Fatores de Impacto*, mesmo que não tenha sido detetada qualquer falha (algo que não acontecia com o *Aliph*). Essencialmente, o ADAPT é composto por três subsistemas: i) o *BFT System* (BFTS), composto por vários protocolos BFT que devem suportar a propriedade de *Abortability*; ii) o *Event System* (ES), responsável por monitorizar o sistema e recolher os valores atuais para os vários fatores de impacto e, iii) o *Quality Control System* (QCS), responsável por analisar estes valores e iniciar uma adaptação quando necessário.

Apesar do uso de aprendizagem automática ser uma contribuição interessante, a solução proposta no ADAPT deixa em aberto três questões importantes que são resolvidas no ByTAM: uma concretização robusta do QCS; uma infraestrutura robusta do ES; e uma maneira simples de criar políticas de adaptação. Em primeiro lugar, embora os autores refiram a possibilidade de desenvolver um QCS tolerante a falhas bizantinas, e apesar de todos os módulos do ADAPT estarem instaladas em todas as réplicas, na concretização proposta a decisão de mudar de protocolo é tomada por uma única réplica (a primária). Consequentemente, o serviço deve confiar na decisão da réplica primária, o que pode comprometer a correção caso esta sofra faltas bizantinas. Em segundo lugar, o monitor apresentado é bastante simples, sendo uma concretização mais robusta protelada para trabalho futuro. Em terceiro lugar, o sistema só suporta políticas baseadas em *funções de impacto* (que catpuram o efeito de cada adaptação), sendo a forma de interpretar estas funções plasmada de forma estática no código, o que restringe a capacidade de concretizar diferentes políticas.

Para além destas questões, a opção seguida pelos autores de colapsar todos estes componentes nas mesmas réplicas (BTFS, ES, QCS) pode afetar negativamente a robustez do sistema, por exemplo, quando se verifica a falha de uma réplica ES e de outra réplica do BFTS, considerando $f = 1$. Isto não ocorre no ByTAM. No nosso caso, cada componente do sistema opera de forma independente, podendo inclusive terem fatores de replicação distintos, como será explicado na Secção 3.6 (ficando ao critério o administrador a decisão de colocar ou não os serviços na mesma réplica).

Justifica-se realçar que, apesar do grande número de projetos de investigação realizados sobre protocolos BFT, estão disponíveis poucas concretizações de código aberto. Neste trabalho, escolhemos usar o BFT-SMaRt[3] que concretiza uma máquina de estados replicada tolerante a falhas bizantinas desenvolvida em Java. O BFT-SMaRt é um sistema modular que concretiza um protocolo BFT semelhante ao PBFT[5] e inclui módulos de transferência de estados e reconfiguração de réplicas. Neste sistema, o processo de reconfiguração é coordenado por uma componente externa, central e confiável (uma das limitações que este artigo pretende superar). A abordagem modular do BFT-SMaRt distancia-se da abordagem monolítica (adotada, por exemplo, pelo PBFT) e fornece também uma interface de programação simples e extensível. O ByTAM uti-

liza o BFT-SMaRt de duas formas: i) como sistema gestor de adaptação automática, e ii) como sistema alvo gerido.

3 ByTAM

Nesta secção apresentaremos o modelo do sistema, sua arquitetura e uma perspetiva geral sobre o modo como os diferentes componentes do ByTAM operam para realizar adaptações de maneira coerente.

3.1 Modelo do Sistema

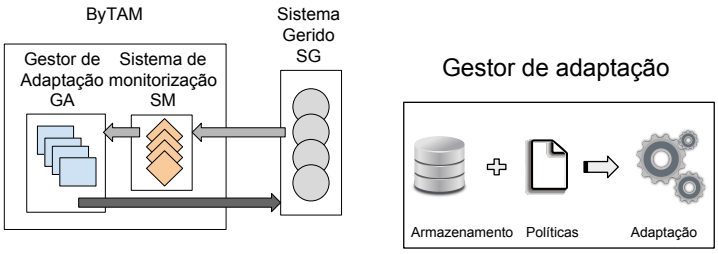
Assumimos o modelo de falhas Bizantino, no qual processos que falham podem comportar-se de maneira arbitrária[10] e a existência de um adversário forte, com capacidade de coordenar processos falhados para comprometer o sistema replicado. No entanto, assumimos que este adversário não pode violar as técnicas de criptografia utilizadas (como MACs, encriptação e assinaturas). Para além disto, assumimos que no máximo podem falhar "f" réplicas de cada componente (seja do gestor da adaptação, seja das réplicas dos sensores e/ou atuadores). Finalmente assumimos uma rede assíncrona onde mais cedo ou mais tarde existem intervalos de sincronia, em que as mensagens são entregues a tempo e o protocolo faz progresso.

3.2 Arquitetura

O ByTAM é composto por dois subsistemas que operam de forma autónoma: o sistema de monitorização (SM) e o gestor de adaptação (GA). Ambos comunicam com o o Sistema Gerido (SG), conforme se ilustra na Figura 1(a). O SG concretiza o serviço fornecido ao utilizador final e, essencialmente, fornece informações sobre as condições operacionais e recebe adaptações, funcionando como um cliente do ByTAM. Um exemplo de um sistema gerido é o DepSpace[2], um espaço de tuplos tolerante a faltas. O SG possui apenas dois requisitos: deve possuir algum mecanismo de reconfiguração (ex. protocolo de mudança de vista ou suportar *Abortability*); e ser monitorizável (exportar métricas operação e desempenho). O SG opera de forma independente dos restantes componentes do sistema, portanto, problemas que impeçam a comunicação com o ByTAM (particionamento da rede, assincronia) têm como efeito apenas atrasos no processo de adaptação para uma configuração ótima. A concretização do SG poderá assumir um modelo de falhas distinto do ByTAM. No entanto, para uma solução globalmente robusta, neste trabalho assumiremos que o SG é também tolerante a falhas bizantinas. Nas próximas secções descreveremos em pormenor o SM e GA, apresentando as suas principais funcionalidades.

3.3 Sistema de Monitorização

O SM consiste numa infraestrutura de sensores que recolhe continuamente dados sobre o ambiente (ex. dispositivos de rede, recursos,...), e do serviço em operação (latência, carga de operações, tamanho das mensagens, etc.), e entrega estes dados ao GA para



(a) Arquitetura do sistema (b) Serviços do GA

Figura 1. ByTAM: Arquitetura e Serviços

armazenamento e processamento. A tolerância a faltas nos sensores é obtida através de replicação de máquina de estados utilizando um protocolo BFT. Assim, pelo menos $3f+1$ réplicas de sensores são necessárias para tolerar f falhas. Desta forma, cada sensor é visto como um cliente da máquina de estados e as atualizações dos parâmetros obtidos pelos sensores são tratados como comandos que são ordenados de forma total pelo protocolo BFT.

Como resultado, os sensores produzem uma sequência linear de valores capturados identificados por tuplos que, para além dos dados recolhidos, incluem o identificador único da réplica/sensor e um número de sequência. Estes tuplos são armazenados no GA. No entanto, os valores de sensores individuais não são usados diretamente. Ao invés disto, o ByTAM espera por um quórum formado por $\lceil \frac{(n+f)}{2} \rceil$ leituras de sensores diferentes. Note-se que, uma vez que as leituras dos sensores são entregues por ordem total, todas as réplicas corretas irão receber a mesma sequência ordenada de valores vindo dos sensores replicados. Assim, uma função determinista pode ser aplicada para eliminar as f leituras mais altas e baixas (extremos) e o valor resultante é utilizado. Uma limitação desta abordagem é a necessidade de executar o consenso para cada leitura de cada sensor. Uma maneira simples de atenuar este custo consiste em agrupar várias leituras e executar o consenso em bloco. Por razões de falta de espaço, omitimos a descrição desta otimização.

3.4 Gestor de Adaptação

O GA é responsável por processar os dados obtidos do SM, registar e avaliar políticas de adaptação e iniciar o processo de adaptação. Para isto é constituído por três serviços, conforme se ilustra na Figura 1(b): armazenamento, gestor de políticas e coordenador de adaptação. Na essência, o GA processa os dados dos sensores, computando métricas de qualidade. Estas métricas são então avaliadas por políticas previamente instaladas. Estas políticas, de acordo com os valores das métricas, podem iniciar uma ação de adaptação. Tal como o SM, o GA é concretizado como um serviço replicado tolerante a falhas bizantinas. Vale a pena salientar que as mensagens entre SM e GA são assinadas digitalmente e, portanto, informações de sensores não autenticados são descartadas.

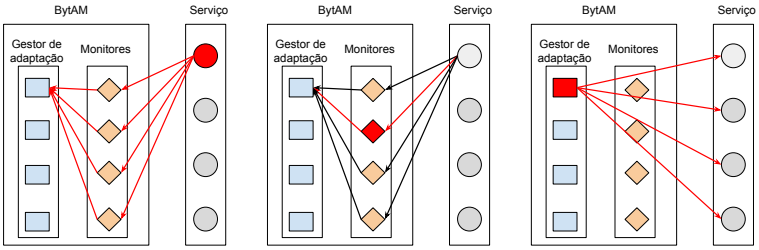
O serviço de armazenamento armazena os dados recolhidos por cada sensor. Conforme explicado na Secção 3.3, o SM produz uma sequência linear de valores, portanto, cada réplica correta do GA possui a mesma sequência de valores em número suficiente para ser processado ou um prefixo desta sequência (e inevitavelmente receberá os restantes valores, pelas propriedades de correção do *consenso*). Assim é possível garantir a tomada de decisões coerentes em todas as réplicas, pois as políticas serão avaliadas sob o mesmo estado do sistema. Ao permitir o registo contínuo da informação enviada pelos sensores, possibilitamos a escrita de políticas que levam em consideração o histórico dos valores recolhidos. Note-se que a informação imediata pode estar sujeita a flutuações transitórias, para as quais o custo de uma adaptação não compensa iniciar uma reconfiguração. Para além disso, torna-se possível identificar padrões de operação ao longo do tempo e adaptar de forma pro-ativa. Apesar do ByTAM ter sido desenvolvido para suportar este tipo de políticas, o desenvolvimento de políticas concretas que consideram relações custo-benefício e tendências é trabalho futuro.

O gestor de políticas controla o ciclo de vida das políticas de adaptação. Isto é, permite listar, instalar/remover políticas, associar/desassociar políticas a um SG específico e ativar/desativar políticas de um dado SG. A definição de políticas de adaptação será pormenorizada na Secção 3.7. Cada política possui um identificador único associado e existe um ficheiro de configuração, que é consultado periodicamente, que indica qual das políticas deve ser executada em cada momento. Isto permite comutar dinamicamente entre várias políticas pré-carregadas. No futuro pretendemos estender o sistema para suportar o carregamento dinâmico de políticas. Este fluxo é suportado por um utilitário que realiza a configuração das políticas de adaptação nas várias réplicas do GA de maneira coerente. Isto confere maior flexibilidade ao ByTAM, quando comparado com as outras soluções, pois oferece uma forma confiável de aplicar novas políticas com o sistema em operação.

O Coordenador de adaptação é o responsável pela configuração de um SG, isto é, de estabelecer um conjunto de parâmetros operacionais iniciais, adicionar/remover réplicas a um grupo SG, definir réplicas ativas e backups, etc. Para além disto, o coordenador de adaptação executa as políticas ativas, suportando diferentes critérios de ativação (periódico, reativo, personalizado) e, caso seja necessário realizar uma adaptação, dispara comandos a fim de reconfigurar o SG.

3.5 Realizando uma Adaptação

Nas secções anteriores descrevemos o funcionamento do GA e como as réplicas tomam as mesmas decisões com base numa representação coerente do estado do sistema. Vale a pena destacar ainda que as réplicas precisam de concordar quando se deve executar uma política, assim como acordar com respeito a qual versão do estado a política deve ser aplicada (de forma a garantir adaptações atempadas e coerentes). Conforme mencionado, uma política pode ser executada em reação a um evento, por exemplo, assim que o valor obtido pelos sensores ultrapassem um determinado limiar definido na política. De qualquer maneira, assumimos que todas as ativações da política são executadas em série. Assumimos também que um número de série é associado a cada ativação da



(a) Falha no SG (b) Falha no SM (c) Falha no GA

Figura 2. Mensagens de réplicas falhadas em diferentes componentes ($f = 1$)

política. Cada ativação da política incrementa este identificador, provocando a disseminação do comando ADAPT$(i+1,s)$ para todas as outras réplicas, onde s é um identificador do estado sob o qual deve ser aplicada a ativação $i+1$ da política. Para além disto, uma réplica que recebe $f+1$ comandos ADAPT de réplicas diferentes, também adota a decisão de ativar a política sobre o estado s, disseminando também o comando ADAPT$(i+1,s)$ para si. Uma réplica do GA executa a política assim que receba $2f+1$ comandos ADAPT iguais. As mensagens entre réplicas do GA são também assinadas digitalmente. Portanto, comandos originados de réplicas não autenticadas não são considerados durante a validação do quórum de mensagens. Por sua vez, caso a política dispare comandos de reconfiguração para o SG, este último só executa a reconfiguração se receber este comando de um quórum de mensagens de réplicas do GA.

3.6 Lidando com Componentes Bizantinos

Conforme explicado anteriormente, os diferentes componentes do ByTAM operam de forma independente, comunicando exclusivamente através da troca de mensagens. Isto facilita a análise do comportamento do sistema quando ocorrem falhas e a concretização de mecanismos que evitem a propagação das mesmas, como se ilustra na Figura 2. Nos próximos parágrafos discutimos os efeitos da ocorrência de faltas em componentes do SG, do SM e GA.

Quando um componente do SG apresenta comportamento bizantino, as leituras dos SM podem obter valores arbitrários ou falhar silenciosamente. No exemplo da Figura 2(a) a réplica do SG pode emitir valores intencionalmente diferentes para cada sensor com o intuito de prejudicar (ou forçar) a ativação de uma determinada política e consequentemente reduzir a disponibilidade do serviço durante este intervalo de reconfiguração. Para evitar que valores arbitrários, produzidos por uma réplica bizantina do SG, sejam propagados até ao GA, cada réplica do SM deve ler o mesmo indicador de todas as réplicas do SG e realizar uma votação tolerante a faltas sobre os valores lidos antes de propagar o resultado da leitura para o GA. Note-se que a maioria das réplicas do SG irá enviar valores coerentes para as réplicas do SM, pelo que todos os SM corretos acabarão por enviar a mesma leitura correta para o GA.

Quando ocorre uma falha nalgum componente do SM, tal como se ilustra na Figura 2(b), este pode não fornecer as leituras ao GA ou alterar arbitrariamente os valores lidos e enviar valores contraditórios para diferentes réplicas do GA. A propagação de valores contraditórios é filtrada pela execução do consenso na entrada do GA. Uma omissão de uma leitura é tratada da mesma forma que a transmissão de um valor errado. Finalmente, os valores errados são recebidos pelo GA mas depois eliminados no processo de votação, que utiliza os valores recebidos das restantes réplicas do SM: ou o valor da leitura é muito maior/menor que os valores produzidos pelas outras réplicas ou encontra-se no intervalo definido pelos valores produzidos por duas réplicas corretas. No primeiro caso, a função determinista elimina os valores extremos durante a votação. No segundo caso, se o valor está no intervalo definido pelas réplicas corretas, então é também um valor correto. Esta estratégia é semelhante à usada noutros contextos, por exemplo em [6].

Finalmente, quando uma réplica do GA falha (Figura 2(c)), ela pode optar por não executar a política ou disparar uma adaptação errada ou num momento inoportuno. No entanto, estes comportamentos serão mascarados pelas restantes réplicas do GA. Note-se que todas as réplicas do GA recebem a mesma sequência de valores enviados pelo SM, conforme descrito anteriormente. Desta forma, cada réplica correta do GA aplica individualmente as políticas e, se for o caso, irá disparar uma adaptação. Uma vez que as políticas são deterministas, todas as réplicas corretas do GA disparam a mesma adaptação. Desta forma, mesmo que uma das réplicas deixe de comunicar com o SG, as restante réplicas do GA acabarão por enviar o comando de reconfiguração, dando início ao processo de adaptação. No segundo caso, se a réplica bizantina enviar um comando indevido para o SG, este não inicia a adaptação de imediato. Ao invés disto, o SG espera um quórum de $\lceil \frac{(n+f)}{2} \rceil$ comandos de reconfiguração de réplicas diferentes para então realizar uma adaptação.

A análise realizada anteriormente pressupõe que as diferentes réplicas de um determinado componente falham de forma independente. Isto obriga a uma seleção criteriosa das máquinas onde as réplicas são executadas. Por outro lado, não existe nenhum inconveniente em co-localizar réplicas de serviços distintos numa mesma máquina (por exemplo, uma réplica do SM e uma réplica do GA) desde que se assegure que a cobertura da hipótese de que não falham mais do que f réplicas de cada serviço é suficientemente alta. Finalmente, os componentes de gestão, como o SM e o GA podem ser partilhados para realizar a adaptação dinâmica de diferentes SGs, uma vez que a frequência das adaptações é tipicamente baixa.

3.7 Políticas de Adaptação

As políticas aceites pelo GA são especificadas na forma *evento-condição- ação* (ECA). Concretamente, para criar uma política é necessário implementar a API do GA e especificar quais os eventos que ativam a política, que condições devem ser verificadas na sua ativação e, finalmente, qual a ação que deve ser executada para aplicar uma adaptação.

Os eventos que ativam a política podem ser de notificação ou temporais. Assim, uma política pode ser registada para ser executada quando é recebido algum valor do SM, usualmente para alguma resposta rápida a um evento definitivo, por exemplo, a

falha por paragem (*crash*) de uma réplica do SG que execute o PBFT (permitindo a reconfiguração do fator de replicação para um número menor de réplicas).

De forma semelhante, recorrendo por exemplo a heurísticas, é possível prever o comportamento de variáveis dinâmicas com base em observações das métricas obtidas anteriormente. Para tanto é possível elaborar uma política ativada periodicamente que selecione um conjunto de leituras obtidas do SM. Por exemplo, é possível utilizar o repositório para atualizar o *conjunto de treino* de um sistema baseado em aprendizagem automática como o ADAPT.

O desenvolvimento de novas políticas passa pela implementação de uma interface *Policy* existente no GA. Esta interface disponibiliza dois métodos: *trigger()* e *execute()*.

No método *trigger()*, são descritos os eventos que estimulam a execução da política ou seja, este método representa uma fase anterior à execução da política onde é averiguado se a sua execução é necessária. Se a execução da política for necessária, então é enviado um comando ADAPT assinado para todas as réplicas do GA. Após a receção de $2f + 1$ comandos ADAPT válidos e de diferentes réplicas, o método *execute()* da política é chamado.

No método *execute()* é criada uma nova configuração que é enviada para o SG. Esta configuração faz uso da API de reconfiguração oferecida pelo SG que está neste momento ativo. O SG ao receber $f + 1$ mensagens válidas e assinadas por diferentes réplicas do GA executa a reconfiguração descrita nessa mensagem.

4 Avaliação

O ByTAM permite reconfigurar qualquer parâmetro disponibilizado pela biblioteca de replicação ativa. Nesta avaliação pretendemos ilustrar a vantagem de possuir um sistema independente e flexível para controlar esta adaptação, ao invés de usar soluções que suportam um conjunto restrito de adaptações embutidas no código. Em particular, recorremos a um cenário onde é mais vantajoso não só alterar o grau de tolerância a faltas mas também um dos parâmetros de configuração do protocolo BFT em funcionamento. Nenhum dos sistemas anteriores oferece a flexibilidade para realizar este tipo de adaptação.

4.1 Concretização

As réplicas e clientes usados na avaliação do ByTAM foram hospedadas no serviço DigitalOcean. Cada réplica/cliente tem o seu próprio ambiente de virtualização (através do KVM). Cada ambiente virtualizado tem acesso a 512Mb de RAM, ligações gigabit ethernet entre os comutadores e ligações a fornecedores de Internet de 10 Gigabit (ethernet), processadores Intel Xeon E5-2630L v2 a 2.40Ghz (6 cores) e discos SSD de 20Gb. Cada réplica tem um IP único. O sistema de adaptação automático é baseado no BFT-SMaRt e, por esse motivo, os componentes são executados numa Java Virtual Machine (JVM). A versão da JVM utilizada foi a 1.8.0_91. Cada réplica foi lançada com um tamanho de heap inicial de 256Mb e com o algoritmo de reciclagem automática de memória G1GC.

Para avaliar o desempenho de cada configuração, submetemos um serviço de BFT a uma carga variável, gerada por máquinas clientes com vários fios de execução. Para

isso recorremos a testes de desempenho oferecidos pelo BFT-SMaRt, nomeadamente o *ThroughputLatencyServer* e o *ThroughputLatencyClient*. Cada pedido ao serviço replicado e cada resposta enviada têm um tamanho de 512 octetos. Cada fio de execução cliente envia 5000 pedidos sequenciais ao serviço replicado sem nenhum tempo de espera entre os mesmos. Durante a fase de calibração das experiências verificámos que uma única máquina a executar clientes não gera carga suficiente para carregar o serviço mas que não são necessárias mais do que duas máquinas para estrangular os servidores. Desta forma, todas as experiências foram feitas recorrendo a duas máquinas cliente, variando o número de fios de execução que se executam em cada uma delas, entre 1 e 15 fios de execução em cada (para um máximo de 30 fios de execução no total).

4.2 Adaptação Avaliada

A adaptação utilizada nesta avaliação corresponde à reconfiguração *scaleDown* da política apresentada na Listagem 1. Esta política descreve duas possíveis respostas do sistema à falha de uma réplica. Se existem máquinas sobressalentes disponíveis no sistema, uma nova máquina é integrada no grupo de servidores. Caso não exista nenhuma máquina disponível de imediato para substituir a máquina falhada, é preferível reconfigurar o sistema de servidores para tolerar uma única falha adicional (mudando a configuração para "$f = 1$" e "$n = 4$"). No entanto, ao realizar a reconfiguração do conjunto de servidores é possível reconfigurar alguns dos parâmetros de configuração do protocolo para otimizar o seu desempenho para essa nova configuração. Em particular, um dos parâmetros usados pela concretização do BFT-SMaRt é a frequência da salvaguarda do estado do histórico. As nossas experiências revelaram que, para a tipologia de carga usada nas experiências, e para uma configuração com "$f = 2$" e "$n = 7$" o valor ideal deste parâmetro é de 1000 enquanto que para uma configuração com "$f = 1$" e "$n = 4$" o valor ideal é de 100 (estes dados foram obtidos experimentalmente, correndo o sistema para diferentes valores deste parâmetro, com intervalos de 50 unidades). O ByTAM, permite reconfigurar o protocolo conjuntamente com a alteração do número de servidores.

4.3 Resultados Obtidos

Aferimos o desempenho do sistema quando, numa configuração inicial com "$f = 2$", "$n = 7$" e limiar de salvaguarda a 1000, ocorre a falha de um servidor. Neste caso, avaliamos três alternativas possíveis: deixar o sistema funcionar com essa configuração até que uma réplica sobressalente possa substituir a réplica falhada (note-se que a configuração tolera ainda mais uma falha); reduzir o número de servidores do serviço para "$f = 1$" e "$n = 4$" mantendo o valor do limiar de salvaguarda usado pelo protocolo (note-se que esta configuração também suporta uma falha adicional); reduzir o número de servidores e ajustar a configuração do limiar de salvaguarda, conforme capturado na política da Listagem 1.

Como métrica de desempenho do sistema usámos o rácio entre o débito observado e a latência de cada uma das operações, em que o débito captura o número de operações realizadas por segundo e a latência o intervalo de tempo entre a execução de um pedido e a receção de uma resposta (em μs). Note-se que, neste tipo de sistema, é geralmente

```
public class AdaptQuorumSize implements Policy {
  private Server server = null;

  @Override
  public void trigger(Events evts) {
    if (evts.has(Event.ReplicaDown)) {
      Server newServer = getBackupServer();
      if (newServer != null) {
        this.server = newServer;
      }
      sendAdaptMsg(); //send notification to GA to run the policy
    }
  }

  @Override
  public void execute() {
    if (server != null) {
      integrate(server);
    } else {
      scaleDown();
    }
  }

  private void scaleDown() {
    Configuration newConfig = new Configuration();
    newConfig.set("f","1");
    newConfig.set("n","4");
    newConfig.set("checkpoint","100");
    sendConfigMsg(newConfig); //send reconfiguration commands to SG
  }
    ...
}
```

Listagem 1. Exemplo de políticas de adaptação do ByTAM.

possível aumentar o débito do sistema à custa de uma aumento da latência observada pelo clientes, pelo que este rácio captura não só a capacidade do sistema mas também a qualidade de serviço oferecida aos clientes.

Os resultados estão apresentados na Figura 3, para diferentes cargas impostas ao servidor. Como se pode ver, o desempenho do sistema melhora sempre após a reconfiguração. Para além disso, ao reduzir o número de servidores, obtém-se melhor desempenho alterando também o limiar usado pelo protocolo para salvaguardar o histórico em disco. Uma grande vantagem do ByTAM, é que a decisão de fazer estas adaptações em conjunto (tal como na política apresentada) ou separadamente, em função de outros critérios, pode ser facilmente aplicada através da re-escrita da política, sem necessitar de realizar nenhuma alteração ao código do gestor da adaptação ou do sistema gerido, ao contrário do que acontecia nos sistemas anteriores.

5 Conclusões

Neste artigo apresentámos o ByTAM, uma arquitetura robusta para realizar a adaptação dinâmica de sistemas tolerantes a falhas Bizantinas. Tanto quanto é do nosso conhecimento, o ByTAM é o primeiro sistema de reconfiguração de código aberto que é tolerante a falhas Bizantinas de todos os componentes do sistema, seja das réplicas do sistema gerido, do gestor da adaptação, ou dos sensores que capturam o estado do sistema. Para além disso, ao contrário das propostas anteriores, o ByTAM suporta a execução de múltiplas políticas de adaptação, sem exigir alterações ao sistema gerido ou ao gestor de adaptação. A versão atual do ByTAM pode ser obtida em `https://github.com/fmrsabino/library/tree/bytam`.

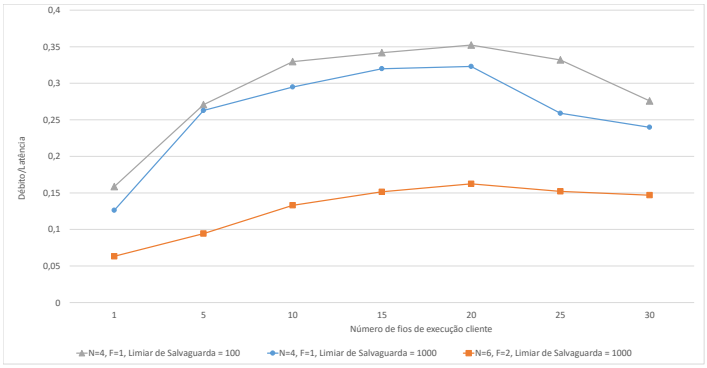


Figura 3. Desempenho das várias configurações

Agradecimentos Este trabalho foi parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) e pelo PIDDAC através dos projetos com as referências PTDC/EEI-SCR/1741/2014 (Abyss) and UID/CEC/50021/2013.

Referências

1. J.-P. Bahsoun, R. Guerraoui, and A. Shoker. Making BFT protocols really adaptive. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 904–913, 2015.
2. A. Bessani, E. Alchieri, M. Correia, and J. Fraga. Depspace: A byzantine fault-tolerant coordination service. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 163–176, 2008.
3. A. Bessani, J. Sousa, and E. Alchieri. 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)*, pages 355–362, 2014.
4. C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to reliable and secure distributed programming*. Springer, 2011.
5. M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *Proceedings of the Operating Systems Design and Implementation (OSDI)*, pages 173–186, 1999.
6. D. Dolev, N. Lynch, S. Pinter, E. Stark, and W. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, May 1986.
7. M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.
8. R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376. ACM, 2010.
9. R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. *ACM SIGOPS Operating Systems Review*, 41(6):45–58, 2007.
10. L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.
11. A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe. BFT protocols under fire. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 189–204, San Francisco, California, 2008.