Monitorização de Sistemas Tolerantes a Faltas Bizantinas para Suportar Adaptação Dinâmica

Bernardo Palma, Daniel Porto and Luís Rodrigues

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

Resumo Os sistemas adaptáveis (SA) mudam o seu comportamento em resposta a variações no ambiente de execução causadas, por exemplo, por faltas ou alterações nos padrões de acesso. Um componente importante em qualquer SA é o sistema de monitorização (SM), responsável por recolher informações sobre a execução e detetar mudanças que justificam a adaptação. O SM é especialmente complexo na presença de faltas Bizantinas, onde elementos do sistema podem produzir mensagens incorretas, que por sua vez podem disparar adaptações indesejáveis, tornando o SA ineficiente ou vulnerável. Neste trabalho, descrevemos i) as escolhas realizadas na construção de um SM robusto e flexível para gerir vários tipos de sensores; ii) os mecanismos de consenso que garantem a obtenção de uma vista coerente do estado do sistema, agregando de forma tolerante a faltas as informações recolhidas pelos sensores. A avaliação demonstra que o nosso SM é facilmente extensível e tem capacidade de escala.

1 Introdução

Os sistemas adaptáveis modificam o seu comportamento em resposta a mudanças nas condições do ambiente de execução que podem ser causadas, por exemplo, por faltas, variações no padrões de acesso durante o dia, utilização de recursos partilhados, carga no sistema imposta pelo número de utilizadores, etc. Estas mudanças dinâmicas provocam alterações que afetam o desempenho dos diferentes componentes do sistema. Por exemplo, alguns protocolos de replicação tolerantes a faltas Bizantinas (TFB) como o Zyzzyva [1] operam sobre condições favoráveis, num modo designado por otimista, onde apresentam o seu melhor desempenho. Entretanto, na presença de faltas, o Zyzzyva precisa executar complexas etapas adicionais de comunicação, que acarretam um maior número de mensagens, reduzindo o seu desempenho [2]. De igual forma, os outros protocolos TFB que se encontram na literatura estão também otimizados para condições específicas de operação [3,4,5,6,7].

A ausência de uma solução padrão, isto é, adequada a todas as condições de operação, criou a oportunidade para construção de sistemas TFB adaptáveis que alternam entre diferentes protocolos em resposta às mudanças dinâmicas no ambiente de execução [6,8,9].

Um componente importante em qualquer sistema adaptável é o sistema de monitorização (SM), que é responsável por recolher informações sobre o funcionamento do sistema, informações estas que podem ser usadas para alimentar as políticas de adaptação. Este componente é particularmente complexo na presença de faltas Bizantinas, onde as réplicas incorretas podem produzir mensagens para ativar adaptações inadequadas, tornando o sistema mais vulnerável a ataques ou passível de apresentar um desempenho subótimo. Para evitar estes problemas, o sistema de monitorização deve também ser resistente a falhas Bizantinas. Isto implica ser capaz de tolerar para além de réplicas incorretas, tolerar também sensores incorretos. Isto é, mesmo que uma fração f dos sensores produzam valores errados, o SM deve conseguir alimentar as políticas com informações exatas e coerentes sobre sistema alvo. Note-se que mesmo as réplicas corretas de um mesmo sensor podem naturalmente divergir nos valores fornecidos. Por exemplo, as leituras podem estar levemente desfasadas no tempo, fazendo com que sensores corretos obtenham leituras com valores distintos. Para além destes desafios, é importante que sistema de monitorização tenha uma arquitetura flexível para permitir o desenvolvimento de novos sensores.

Neste trabalho apresentamos MonBiz, um SM autónomo, robusto e flexível que permite a criação e gestão de diversos tipos de sensores, agrupando informações de diversas fontes para exportar uma visão coerente do estado do sistema, mesmo na presença de faltas Bizantinas. O remanescente deste artigo está organizado da seguinte forma: na Secção 2, apresentamos o contexto em que se insere o nosso trabalho, com foco particular na estratégia de monitorização dos protocolos adaptáveis. Na Secção 3 apresentamos o modelo do sistema e as premissas que temos em consideração. Na Secção 4 descrevemos a visão geral, detalhando a arquitetura, interfaces e principais funcionalidades bem como as escolhas realizadas no desenho do sistema. Na Secção 5, identificamos os parâmetros quantitativos e qualitativos e mostramos os resultados da avaliação do sistema. Por fim, as conclusões são apresentadas na Secção 6.

2 Trabalho Relacionado

Existem na literatura diversos sistemas de monitorização desenvolvidos especificamente para detetar falhas, como o Falcon[10], Pigeon[11] e Albatross[12]. Estes sistemas foram desenvolvidos com um modelo de falhas por paragem, e portanto, não são adequados para um ambiente suscetível a faltas Bizantinas. Naturalmente, possuem similaridades com o MonBiz. Por exemplo, o Falcon disponibiliza uma API para desenvolvimento de sensores, mas diferentemente do MonBiz, pode terminar processos para garantir coerência das informações. O Pigeon é capaz de detetar também falhas na rede e ao contrário do Falcon não mata processos, porém não é capaz de garantir precisão da informação quando a falha é detetada. O Albatross opera em ambientes em que existe um gestor de SDN, ele também atua para garantir coerência das informações mas diferentemente do Falcon, desconecta processos ao invés de mata-los. Otimizações propostas nestes sistemas poderão ser exploradas em futuras versões do MonBiz.

Detetor de faltas Bizantinas (DFBs) [13] é uma abstração que permite construir protocolos TFB a partir de elementos que dão indicações não confiáveis sobre faltas em processos que, respeitando algumas premissas, podem resolver o consenso. Um tipo de DFB proposto em [14] para redes sem-fio, usa a ordem lógica em que os processos entregam mensagens, observando o padrão de comunicação durante a execução do protocolo para detetar faltas. O nosso sistema não propõe novas primitivas para resolver o consenso. Ao invés disto, utiliza o consenso para estabelecer uma visão coerente de diversos sensores para, além de detetar falhas, monitorizar alterações no ambiente de execução que justifiquem uma adaptação.

Alguns protocolos adaptáveis TFB como o *Aliph*[6], ADAPT[8] e *Bytam*[9], têm no seu núcleo um componente de monitorização, responsável por detetar e notificar falhas ou eventos importantes em componentes do sistema ou na rede, e desta forma despoletar adaptações.

O *Aliph* é um dos primeiros protocolos TFB capaz de suportar adaptação dinâmica. Em essência, o sistema alterna por uma ordem fixa entre três protocolos diferentes, nomeadamente *Quorum*, *Chain* e *PBFT*. A troca entre estes protocolos é definida por condições definidas de forma estática no código, que dependem da operação do protocolo em execução, não existindo um sistema de monitorização explicita do ambiente de execução.

O ADAPT já recorre a um sistema de monitorização, designado por Sistema de Eventos (SE), para alimentar técnicas de aprendizagem automática que são, por sua vez, responsáveis por escolher, em cada momento, a melhor configuração para o sistema. A concretização do SE é bastante simples, limitando-se a ler informações da aplicação para obter parâmetros, tais como o tamanho das mensagens e o número de clientes, não concretizando nenhuma técnica de tolerância a faltas (e como tal, não sendo robusto na presença de sensores Bizantinos).

O Bytam, apresenta maior flexibilidade comparativamente ao *Aliph* e ao ADAPT para o desenvolvimento de políticas de adaptação, permitindo defini-las de maneira mais geral, na forma de *Evento-Condição-Ação*. Além disto, contrastando com o ADAPT, o Bytam possui um conjunto de monitores concretizados de maneira robusta, sendo capaz de tolerar faltas Bizantinas. No entanto, a infraestrutura sensores que compõe o módulo de monitorização ainda é demasiado rígida, restringido-se a um conjunto fixo de sensores desenvolvidos de forma integrada ao sistema, não sendo simples estender para adicionar novos sensores.

O MonBiz tem por objetivo superar estas limitações, fornecendo uma infraestrutura de monitores flexível, que permite comunicação com sensores simples e replicados; extensível, tornando mais simples a adição de novos sensores através da implementação de uma API; e robusta sendo capaz de tolerar falhas no agregador e sensores replicados.

3 Modelo do Sistema

Assumimos um sistema distribuído composto por vários processos que comunicam por troca de mensagens. O sistema é assíncrono, no sentido em que

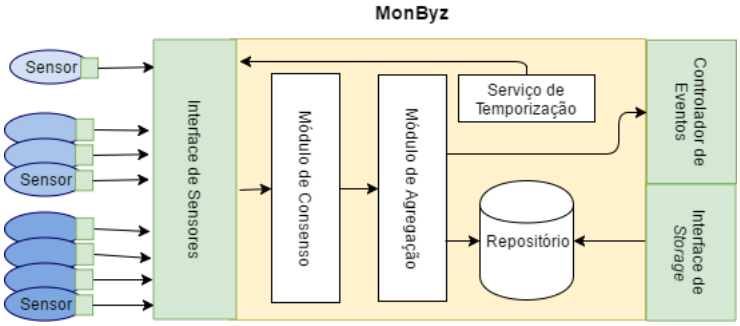

Figura 1. Arquitetura do sistema

os tempos de processamento e de comunicação podem ocasionalmente exceder qualquer limite que se tente impor previamente. No entanto, assumimos que uma maioria dos processos, incluindo aqueles que executam sensores, tem um comportamento no tempo que não inviabiliza a adaptação atempada do sistema. Os processos têm também acesso a uma fonte externa de tempo. Esta fonte é usada pelos sensores para estampar as leituras com uma marca temporal; assumimos que a sincronização dos relógios é externa ao nosso sistema e é realizada por uma fonte confiável.

Os diversos componentes do sistema estão sujeitos a faltas Bizantinas, podendo exibir comportamentos arbitrários. Estas faltas podem ter causas naturais ou resultar da intrusão de agentes maliciosos. Assumimos que no máximo $f = \lfloor \frac{n+1}{3} \rfloor$ réplicas de cada componente podem exibir um comportamento Bizantino, em que n é o número total de réplicas; estas réplicas podem entrar em conluio para tentar subverter o sistema. Assumimos que os processos possuem recursos limitados e não conseguem quebrar as técnicas de criptografia utilizadas nos nossos algoritmos.

4 Arquitetura

O sistema de monitorização MonBiz é composto por três tipos de componentes principais apresentadas na Fig. 1, nomeadamente: um *Agregador*, um *Repositório*, um conjunto de *Sensores*, e um serviço de temporização.

Os sensores recolhem informações sobre o sistema a adaptar (SA) e sobre o ambiente de execução. Os sensores suportam dois modos distintos de operação: podem reagir a eventos esporádicos, por exemplo, falhas, ou fornecem um fluxo de atualizações sobre alguma característica do SA, por exemplo, fornecer leituras periódicas da carga do sistema. Para monitorizar um sistema de forma tolerante a faltas é natural a utilização de múltiplos sensores, cujos valores podem ser comparados e/ou agregados, para desta forma obter maior robustez. Consequentemente podem ser geradas notificações replicadas para um mesmo evento, provenientes de diversos sensores. Um sensor com defeito pode reportar

Tabela 1. Propriedades do sensores

Nome	identifica sensores simples ou grupos de sensores (réplicas) de forma única
Identificador	distingue entre as diferentes réplicas de um sensor (quando aplicável)
Cardinalidade	tamanho do grupo sensores
Faltas	limite de falhas com qual o sensor (ou grupo) consegue operar
Modo de operação	reativo ou periódico
Tipo	Bizantino, *crash*, não-confiável
Credenciais	chaves para autenticar a origem dos valores transmitidos

eventos que não ocorreram, omitir os que aconteceram ou fornecer uma descrição errada sobre os mesmos.

Para além disso, assumimos que o sistema gerido é tolerante a faltas e, como tal, possui várias réplicas que precisam de ser monitorizadas. Cada sensor pode monitorizar uma ou mais réplicas, e cada réplica deve ser monitorizada por pelo menos um sensor.

O MonBiz recolhe os dados de todos os sensores num serviço *Agregador*, que usa os valores fornecidos por diferentes réplicas para filtrar os dados incoerentes produzidos pelos sensores defeituosos. Finalmente, os dados já processados são armazenados de maneira coerente e robusta num *Repositório*, para permitir a consulta por parte de outros serviços. Uma descrição pormenorizada deste componentes será fornecida nas próximas secções.

4.1 Sensores

Os sensores têm a função de recolher informações do SM e do ambiente de execução. O MonBiz permite configurar um sensor para tolerar diferentes tipos de faltas, nomeadamente é possível configurar um sensor como não tolerante a faltas, tolerante a faltas por paragem ou tolerante a faltas Bizantinas. O tipo de configuração escolhida depende de vários fatores, como a semântica do valor a recolher, da possibilidade de instalar ou não sensores independentes para observar um dado fenómeno. Por exemplo, o débito do sistema ou número de clientes ativos podem ser observados por diferentes réplicas e, como tal, ser lidos por sensores independentes, o que permite ter um sensor tolerante a faltas Bizantinas para estas métricas. Por outro lado, alguns sensores possuem características que não permitem a sua replicação ou cuja replicação não faz sentido (por exemplo, estatísticas da CPU de um determinado servidor).

No MonBiz os sensores recolhem dados do sistema através de sondas, de forma passiva, ou podem ainda encapsular sensores legados, enviado esse mesmos dados ao *Agregador*. Desta forma assemelham-se a clientes que enviam comandos para a máquina de estados replicada (MER). Entretanto, não precisam de esperar por uma resposta do *Agregador*, visto que sensores não executam ações sob o sistema. Esta restrição é intencional, visto que adaptações eficientes requerem conhecimento das estratégias do SA, e por isto entendemos que devem ser sistematizadas e coordenadas por um serviço específico. As propriedades dos sensores no MonBiz, são descritas na Tabela 1.

Cada sensor envia ao serviço Agregador uma mensagem contendo a informação recolhida associada a um número de sequência atribuído localmente e incremen-

Listagem 1.1. Classe PeriodicSensor

```
1 package argus.sensors;
2 ...
3 public abstract class PeriodicSensor extends BaseSensor{
4 public PeriodicSensor(Integer id, String type, PrivateKey pKey){...}
5 public abstract SignedMessage collectValue();
6 }
```

Listagem 1.2. Exemplo de configuração do Agregador

```
1 identifier=Throughput
2 sensorType=Metric
3 f=1
4 quorum=3
5 aggregationFunction=argus.aggregator.function.IntAverageFunction
```

tado a cada nova mensagem. Para além disto, a mensagem também deve conter o nome e identificador do sensor e um registo de tempo indicando quando a leitura foi feita. Desta forma, sensores replicados podem ser agrupados no Agregador como um único sensor virtual. No caso de sensores tolerantes a falhas Bizantinas, a mensagem enviada ao Agregador deve ser assinada por cada sensor.

Classificamos a informação recolhida em dois tipos: métricas e eventos. Definimos as métricas como valores numéricos que representam a informação recolhida (e.g., a carga média da CPU de uma dada réplica). Por outro lado, classificamos como eventos, a ocorrência de situações como mudança de líder ou a deteção de falha em uma réplica.

À medida que os sistemas evoluem, torna-se necessário monitorizar um conjunto adicional ou diferente de parâmetros. Desta forma, o MonBiz deve ser capaz de expandir sua capacidade de monitorização. De forma a facilitar o desenvolvimento de sensores, disponibilizamos algumas interfaces em que, por exemplo, apenas é necessário implementar o método de coleção, e.g Listagem 1.1. Para além disto, adaptações como recolocação ou troca de réplicas, quando realizadas, podem desligar ou instalar sensores em conjunto, e como resultado possivelmente mudar a configuração do grupo de sensores. Portanto, o Agregador disponibiliza uma interface que permite o gerência de novos sensores, para desconectar antigos, adicionar e remover réplicas, de maneira atómica e integrada mudar as funções de agregação aplicadas aos sensores. No momento da inicialização o sistema, também permite registar/carregar automaticamente as informações referentes aos sensores existentes, bastando defini-las num ficheiro chamado *aggregation.config* (e.g Listagem 1.2) para cada sensor, juntamente com uma diretoria contendo as chaves públicas pertencentes ao mesmo.

4.2 Agregador

Para simplificar a implementação dos sensores e conferir robustez, o *Agregador* foi desenvolvido como um serviço replicado, estendendo uma biblioteca

para implementação de máquinas de estados replicada tolerante a falhas bizantinas bastante popular denominada BFTSMaRt[15]. Assim, os valores recolhidos através de mensagens dos sensores são ordenados pelo protocolo TFB subjacente. No final do mesmo, todas as réplicas corretas do agregador concordam com a ordem e o conjunto de valores recolhidos dos sensores distintos. Uma vez recolhida uma quantidade mínima de mensagens e estabelecida uma visão coerente dos valores recebidos no agregador, é aplicada uma função de agregação para uniformizar os valores recebidos.

Cada sensor replicado possui uma função de agregação determinista que é aplicada quando a visão dos sensores sobre um determinado evento atinge consenso. Isto permite ao agregador eliminar valores extremos, convergindo a visão dos sensores num único representante da métrica/evento, bem como decidir o respetivo instante associado. Esta função é aplicada pelo módulo de Agregação apresentado na Fig. 1. Um exemplo simples desta função de agregação é computar uma média eliminando as valores mais baixos e mais altos de f sensores, ou ordena-los e selecionar o valor central. Uma vez que o protocolo TFB garante que cada réplica correta do agregador decide pelos mesmos valores obtidos pelos sensores na mesma ordem e a função de agregação é determinista, todas as réplicas corretas obtém o mesmo valor agregado.

Embora não seja estritamente necessário agregar dados recolhidos por sensores não-replicados, a função de agregação pode servir para realizar um pré-processamento do valor recebido, por exemplo, aumentando a sensibilidade do sensor multiplicando o valor recebido por um fator, ou agrupar valores de sensores com frequência mais alta para aliviar o custo no processamento.

4.3 Repositório

Embora as adaptações possam ser iniciadas em resposta a eventos imediatos ou recentes, alguns sistemas realizam adaptações pro-ativas, com base em previsões do comportamento do sistema a partir da observação de acontecimentos ao longo do tempo. Para permitir este tipo de adaptação, o MonBiz regista os dados recolhidos pelos sensores num repositório e disponibiliza os mesmos, já reconciliados, através da interface de *armazenamento*, que pode ser acedida por outros sistemas, como por exemplo um gestor de políticas. O protótipo do serviço de repositório foi concretizado utilizando o H2 [16], uma base de dados relacional. Para garantir a coerência das consultas ao repositório por diferentes réplicas do gestor da adaptação, cada entrada é marcada com um identificador numérico único. Este funciona como o número de versão do repositório, podendo ser usado para limitar a consulta até uma determinada versão/entrada.

4.4 Controlador de Eventos

Apesar de também serem guardados no repositório, eventos são um tipo de informação que por norma requer uma reação imediata. Como tal, aquando da sua ocorrência o sistema alimentado pelo MonBiz poderá querer despoletar uma ação relacionada com o mesmo. Para isso, o nosso sistema oferece a possibilidade

de registar *handlers* associados a determinados eventos. Quando o evento é capturado pelos sensores e devidamente agregado, o respetivo *handler* é chamado podendo, por exemplo, meramente informar o outro sistema da sua ocorrência.

4.5 Serviço de Temporização

O MonBiz oferece também um simples serviço de temporização que pode ser útil para notificar, por exemplo, um gestor de adaptação que um certo período de carência terá terminado ou que estará na altura de reavaliar as suas políticas. Porém, esta notificação só acontece quando o temporizador for despoletado numa maioria das replicas do sistema e passado pelo consenso do mesmo. Os temporizadores funcionam como clientes da maquina de estados replicada, usando um cliente interno ao MonBiz. Estes enviam os eventos à medida que são despoletados, e tal como acontece com os sensores, estes valores passam pelo agregador onde uma acumulação acontece. Finalmente quando o *quorum* é atingido, o evento pode então ser entregue. O serviço disponibiliza dois tipos de temporizadores: Temporizadores periódicos (que, como o nome indica, são despoletados repetidamente com um período de tempo definido no momento do seu registo) e temporizadores que apenas são despoletados uma vez, com um atraso também ele definido no momento do seu registo.

4.6 Módulo de Consenso

Conforme mencionado, o agregador agrupa os valores dos sensores para garantir que todas as réplicas corretas executam operações sobre a mesma visão coerente dos dados. As mensagens de sensores são relacionadas de acordo com as propriedades especificadas para o sensor ou grupo de sensores replicados, definidas no agregador. De maneira semelhante, é definido o limite mínimo para a quantidade de mensagens que cada tipo de sensor deve obter com base na quantidade de faltas toleradas, cardinalidade e tipo de sensor, mostrado na secção 4.1. As mensagens dos sensores podem ser ordenadas à medida que chegam ou acumuladas para posterior ordenação. Experimentámos três abordagens para concretizar este módulo e reportamos o seu comportamento na Secção **??**.

Na primeira concretização, denominada CPré, cada valor enviado pelos sensores é totalmente ordenado, através do consenso. Depois é entregue ao agregador para acumulação e eventual agregação após ser atingido o *quorum* definido para o respetivo sensor. Isto significa que mesmo para sensores replicados, um consenso é corrido para cada valor recebido (ignorando possível *batching*). Aquando da entrega do valor ordenado ao agregador, a validade do mesmo é testada pela seguinte ordem: primeiro, verifica-se que existe um sensor com o mesmo nome do valor recebido registado no sistema; depois, valida-se a autenticidade do valor, verificando que foi corretamente assinado pelo sensor correspondente ou uma da suas réplicas; posteriormente, verifica-se a frescura do valor, nomeadamente se o número de sequência associado ainda não foi acumulado; por último verifica-se se o *quorum* foi atingido.

As duas restantes concretizações, CPós-Total e CPós-Dispersa, executam a acumulação antes da ordenação. Assim sendo, nestas os sensores enviam os seus valores diretamente para o agregador para serem acumulados, sem ordem acordada. Durante esta acumulação, cada um dos valores passa pela verificação descrita acima. Quando o *quorum* é atingido, a acumulação é enviada por um cliente interno para ser totalmente ordenado, sendo finalmente corrido o consenso. Esta escolha de concretização adiciona, portanto, um passo extra de comunicação ao protocolo. Após o conjunto de valores ser ordenado é entregue novamente ao agregador para poder finalmente ocorrer a agregação, passando primeiro por um passo de validação semelhante ao anterior.

Na variante CPós-Total, todas as réplicas corretas enviam o conjunto de valores acumulados assim que atingem o *quorum* para serem totalmente ordenados. Como tal, as acumulações são efetivamente replicadas, não sendo necessário uma réplica líder. Assim que uma destas mensagens passa na verificação final, após a ordenação, as restantes são descartadas. Isto significa que o número de consensos que precisa de ser executado não difere do CPré, apresentando assim um custo claro. Mesmo no melhor caso, esta desperdiça trabalho feito, devido à replicação das mensagens enviadas para o consenso mas, por outro lado, mostra uma maneira simples de concretizar acumulação pré-consenso, sem alterar a máquina de estados replicados e sem necessitar de explicitamente tratar de possíveis falhas ocorridas.

A variante CPós-Dispersa tenta melhorar a anterior. Nomeadamente tenta evitar correr um consenso para cada valor recebido. Para isso, o envio das acumulação é distribuído entre as réplicas, aplicando uma função de *hash* ao nome único do sensor. Assim, cada um dos sensores é atribuído a uma determinada réplica. No caso ótimo, sem a ocorrência de falhas bizantinas, cada réplica só se encarrega de enviar as acumulações dos sensores que lhe compete. Com esta otimização a carga é, portanto, distribuída entre as réplicas. Porém, existe a possibilidade da propriedade de progresso de determinados sensores replicados ser afetada quando uma das réplicas se torna bizantina, quer por enviar valores errados quer por não os enviar. Para lidar com estes casos, quando as réplicas corretas detetam falta de progresso com um determinado sensor, revertem para a primeira concretização, CPós-Total, para os sensores atribuídos à réplica bizantina.

5 Avaliação

A avaliação apresenta uma análise comparativa do desempenho das diferentes concretizações propostas. Uma discussão das qualidades da moldura proposta, nomeadamente da facilidade com que permite desenvolver novos sensores, é omitida por falta de espaço, podendo ser encontrada em [17].

As réplicas do sistema e sensores foram lançados em ambientes virtualizados hospedados pelo serviço *DigitalOcean*, cada um lançado no seu ambiente individual. As máquinas virtual tem acesso a 4 cores de CPU virtualizados, 8GBs de RAM e com um disco SSD de 80GBs cada. Tendo em conta que o nosso sis-

tema foi desenvolvido com recurso à biblioteca Bft-SMaRT que foi desenvolvida em Java, utilizamos a versão 1.8.0_131 para correr o nosso sistema, mantendo os limites por omissão da *heap*. De forma a avaliar o desempenho das diferentes concretizações, estas foram sujeitas a diferentes níveis de carga. Nas experiências, um valor é considerado decidido a seguir a ser agregado, como tal, de forma a que os resultados não sejam afetados pela complexidade da função de agregação, utilizamos uma que apenas retorna o primeiro valor do conjunto obtido. Para além do mais, os valores decididos não são guardados no repositório para testar apenas a capacidade do algoritmo. Tendo em conta que queremos analisar os limites de cada concretização com acumulação prévia antes do consenso e compara-la com a implementação base em que a acumulação é feita posteriormente ao consenso, os testes serão sintéticos na carga que vão gerar, não representando portanto cenários de uso reais. Na instalação do agregador, definimos $f = 1$ gerando, portanto, 4 réplicas do mesmo.

O sensor utilizado é baseado no *microbenchmark* já oferecido pela biblioteca de MER. Cada sensor é lançado como um fio de execução independente que envia sempre o mesmo valor corretamente assinado, sem que ocorra nenhuma espera de forma a atingir um nível elevado de carga. Este valor é enviado um total de 10000 vezes por cada sensor. Nesta avaliação, alteramos a quantidade dos sensores que são lançados e alternamos entre replicados e não replicados. Nos testes com sensores não replicados em cada máquina cliente são lançado 16 sensores em simultâneo, enquanto que nos replicados são apenas lançados 4 sensores. Note-se porém que cada um destes últimos são na realidade 4 fios de execução diferentes, gerando portanto um nível de carga equivalente entre os dois tipos de maquinas cliente. Em ambos os testes, com sensores não replicados e replicados, executamos seis experiências cada, aumentando o número de máquinas cliente de 1 até 6, perfazendo então 16, 32, 48, 64, 80, 96 e 4, 8, 12, 16, 20, 24 sensores para os testes não replicados e replicados, respetivamente. Os valores de débito foram obtidos fazendo a mediana dos valores máximos atingidos entre as 4 réplicas. Para além do mais, é importante referir que a função de dispersão da concretização CPós-Dispersa distribui os sensores de forma uniforme entre as diferentes réplicas.

Os resultados das nossas experiências com sensores não replicados são apresentados na Figura 2(a). Como esperado, a variante CPós-Total apenas conseguiu manter um desempenho semelhante às restantes variantes nas duas primeiras experiências, começando o seu débito a cair daí em diante. Salienta-se também que a partir da segunda experiência, inclusive, não foi capaz de concluir as mesmas, ocorrendo sempre uma exceção por esgotar a memória *heap* do Java. Isto acontece devido à ineficiência desta implementação, visto gerar uma quantidade de mensagens superior às restantes, bem como apresentar uma maior carga de trabalho, em parte devido também ao número de mensagens. Em relação às restantes, podemos verificar que atingem desempenhos semelhantes na maioria das experiências. Apenas na última se nota diferença considerável em que a variante CPós-Dispersa aparenta ter atingido um possível máximo, e a variante CPré mostra conseguir processar cerca de 200 operações a mais.

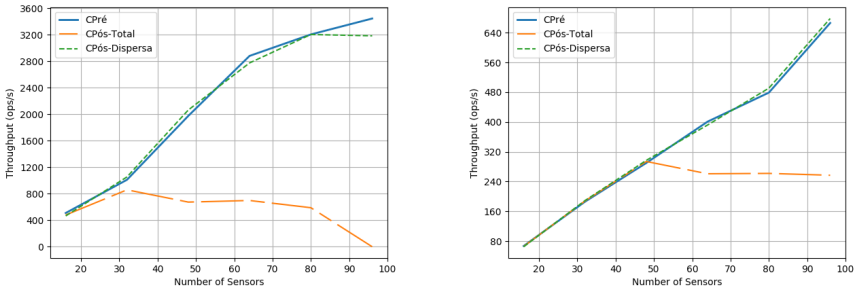

Figura 2. Desempenho das três variantes, sem e com replicação dos sensores.

Nos resultados com sensores replicados, apresentados na Figura 2(b) e salientando novamente que quatro sensores equivalem a um replicado, o comportamento repete-se, no sentido que a variante CPós-Total mais uma vez apresenta o pior desempenho. Porém desta vez podemos ver que a sua capacidade de lidar com as mensagens dos sensores apenas começa a cair a partir dos 12 sensores replicados que corresponde na realidade a uma carga de mensagens equivalente a 48 sensores não replicados. Este mais uma vez não consegue completar todas as experiências, neste caso as últimas três. Com as variantes CPré e CPós-Dispersa não existem disparidades significantes no desempenho para as cargas testadas, como se pode constatar.

De forma geral, conseguimos ver que as variantes CPré e CPós-Dispersa mostram resultados coerentes e comparáveis, apesar da primeira conseguir obter melhor desempenho com sensores não replicados. Isto significa que existe algum mérito na variante CPós-Dispersa, permitindo-nos também argumentar que uma solução mais integrada, nomeadamente entre o agregador e a biblioteca Bft-SMaRT poderia ter um desempenho semelhante ou superior.

6 Conclusão

Neste artigo apresentámos a arquitetura de um sistema de monitorização tolerante a falhas bizantinas com o objetivo de alimentar um gestor de adaptação. Para além disso apresentamos também algumas das interfaces que permitem estender a infraestrutura de sensores do sistema. Tanto quando é do nosso conhecimento, este é o único sistema de monitorização tolerante a falhas bizantinas e que considera sensores com diferentes modelos de falha, incluindo faltas por paragem e Bizantinas.

Agradecimentos: Este trabalho for parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) através dos projectos com referências PTDC/ EEI-SCR/ 1741/ 2014 (Abyss) e UID/ CEC/ 50021/ 2013.

Referências

1. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: Speculative byzantine fault tolerance. In: Proc. of the 21st ACM SIGOPS symposium on Operating systems principles (SOSP), Stevenson (WA), USA (2007) 45–58
2. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In: Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI), Boston (MA), USA (2009) 153–168
3. Castro, M., Liskov, B.: Practical Byzantine Fault Tolerance. In: Proc. of the 3rd Symposium on Operating System Design and Implementation (OSDI), New Orleans (LA), USA (1999) 1–14
4. Abd-El-Malek, M., Ganger, G., Goodson, G., Reiter, M., Wylie, J.: Fault-scalable Byzantine fault-tolerant services. In: Proc. of the 20th ACM Symposium on Operating Systems Principles (SOSP), Brighton, United Kingdom (2005)
5. Cowling, J., Myers, D., Liskov, B., Rodrigues, R., Shrira, L.: HQ Replication: a hybrid quorum protocol for byzantine fault tolerance. In: Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI). (2006)
6. Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M., Aublin, P.l., Lyon, I.: The Next 700 BFT Protocols. In: Proc. of the 5th European Conference on Computer Systems (EuroSys), Paris, France (2010)
7. Amir, Y., Coan, B., Kirsch, J., Lane, J.: Prime: Byzantine replication under attack. IEEE Transactions on Dependable and Secure Computing **8**(4) (2011) 564–577
8. Bahsoun, J., Guerraoui, R., Shoker, A.: Making BFT Protocols Really Adaptive. In: Proc. of IEEE 29th Int'l Parallel and Distributed Processing Symposium (IPDPS), Hyderabad, India (2015)
9. Sabino, F., Porto, D., Rodrigues, L.: Bytam: um gestor de adaptação tolerante a falhas bizantinas. In: Actas do 8º Simpósio de Informática (Inforum), Lisboa, Portugal (2016)
10. Leners, J.B., Hung, W.l., Aguilera, M.K., Walfish, M.: Detecting failures in distributed systems with the FALCON spy network. In: Proc. of the 23rd ACM Symposium on Operating Systems Principles (SOSP), Cascais, Portugal (2011)
11. Leners, J.B., Gupta, T., Aguilera, M.K., Walfish, M.: Improving availability in distributed systems with failure informers. In: Proc. of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI), Lombard (IL), USA (2013) 427–442
12. Leners, J.B., Gupta, T., Aguilera, M.K., Walfish, M.: Taming uncertainty in distributed systems with help from the network. In: Proc. of the 10th European Conference on Computer Systems (EuroSys), Bordeaux, France (2015)
13. Malkhi, D., Reiter, M.: Unreliable intrusion detection in distributed computations. In: Proc. of the 10th IEEE Workshop on Computer Security Foundations, Rockport (MA), USA (1997)
14. De Lima, M.S., Greve, F., Arantes, L., Sens, P.: The time-free approach to Byzantine failure detection in dynamic networks. In: Proc. of the 41st Int'l Conference on Dependable Systems and Networks (DSN), Hong Kong, China (2011)
15. Sousa, J., Bessani, A.: From Byzantine consensus to BFT state machine replication: A latency-optimal transformation. In: Proc. of the 9th European Dependable Computing Conference (EDCC), Sibiu, Romania (2012)
16. Mueller, T.: H2 Database Engine. http://www.h2database.com
17. Palma, B.: Monitoring Byzantine Fault Tolerant Systems. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa (2017)