WA3: Auditoria a Sistema de Leilões com BFT-SMaRt integrado

Jorge Pereira, Pedro Camponês, Rafael Sequeira
DI
FCT-UNL
Lisboa, Portugal
{jff.pereira, p.campones, rm.sequeira}@campus.fct.unl.pt

***Abstract*—BFT-SMaRt[1] is a program that handles state machine replication whilst withstanding Biznatine faults. BFT-SMaRt's design priority is its modularity[2], allowing it to be modified and easilly integrated within a system. In order to attest its use, an auction system that uses BFT-SMaRt has been implemented.**

In this report, the system's security, fault tolerance, and performance are tested. Based on the results of these tests, it's possible to attest the successful integration of BFT-SMaRt.

To conclude the report, an improvement of the auction system is presented. This improvement contemplates the possibility that a client might contact a faulty replica; a case that is not supported by the current system.

***Abstract*—BFT-SMaRt[1] é um programa que lida com replicação de máquinas de estado e suporta falhas Bizantinas. A prioridade do design do BFT-SMaRt é a sua modularidade[2], permitindo que este seja modificado e facilmente integrado em um sistema. De forma a provar o seu uso, um sistema de leilões que utiliza BFT-SMaRt foi implementado.**

No relatório apresentado, a segurança, tolerancia a falhas e performance do sistema são testadas. Com base nos resultados dos testes, comprovou-se o sucesso da integração do BFT-SMaRt.

Para concluir o relatório, é apresentado uma melhoria ao sistema de leilões. Esta melhoria contempla a possibilidade do cliente contactar um réplica com falhas; uma situação que não é suportada pelo sistema atual.

I. Introdução

Qualquer sistema é susceptível a falhas, sendo estas mais propicias em proporção à dimensão do sistema; quer em quantidade de componentes, a complexidade mecânica destes e a complexidade lógica das operações que estes executam. Um sistema distribuído é caracterizado por uma multiplicidade de componentes que comunicam entre si através de mecanismos complexos.

É possível mitigar o surgimento de falhas em sistemas, não obstante, é impossível impedir a eventual ocorrência destas. Deste modo é necessária a criação de sistemas que pressuponham a existência de falhas e implementem mecanismos para as tolerar.

Uma falha pode manifestar-se através de um *crash* de um componente, como resultado, este pára de interagir com o restante sistema que integra; contudo, uma falha também se pode manifestar pelo comportamento arbitrário de um componente, sendo este comportamento o resultado de uma falha de hardware, erros de software ou uma intrusão por parte de um adversário. O comportamento arbitrário apresentado por parte de um componente é designado uma falha *Bizantina*.

O programa *BFT-SMaRt*[1] é uma implementação de um sistema de replicação de máquinas de estado com consistência forte, que tolera falhas Bizantinas e *crash* de componentes.

Na implementação de um sistema de leilões, o BFT-SMaRt foi integrado para gerir a replicação dos dados e tolerância a falhas. O relatório apresentado apresenta uma auditoria da integração do BFT-SMaRt neste sistema.

As secções II, III e IV abordam, respectivamente, o modelo de sistema, do adversário e o modelo de falhas. Através dos dados apresentados nestes modelos, estabelece-se as propriedades do sistema a sob auditoria, e as condições nas quais esta decorre.

Na secção V é apresentado a preparação do sistema para a realização da auditoria, cujas experiências são apresentados nas secções VI, VII e VIII; com cada secção apresentando os resultados da auditoria nas vertentes de segurança, tolerância a falhas e performance, respectivamente.

O relatório é concluído na secção IX, sendo apresentado uma conclusão do trabalho realizado e propondo-se melhorias futuras.

II. Modelo do Sistema

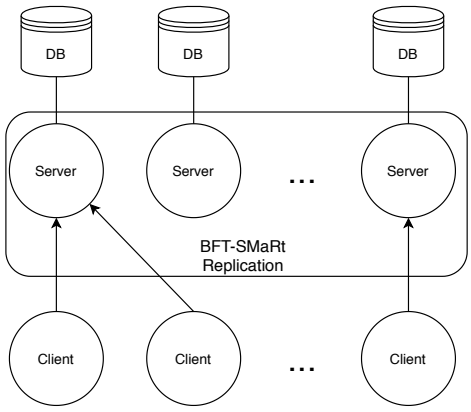

Fig. 1. Modelo de sistema: Serviço de leilões com BFT-SMaRt integrado

O sistema, como apresentado na Fig. 1, é composto por um conjunto limitado de servidores aplicacionais que podem estar fisicamente isolados entre si e nos quais está integrado o programa BFT-SMaRt. A cada servidor aplicacional corresponde uma base de dados *MySQL*. O estado de cada base de dados é idêntico às restantes, uma propriedade garantida pela replicação com consistência forte assegurada pelo BFT-SMaRt.

O sistema suporta um número arbitrário e não fixo de clientes, sendo que estes comunicam apenas com um único servidor através de operações *HTTPS*, que podem consultar parte do estado das réplicas, assim como fazer alterações.

Quando um servidor recebe uma operação numa mensagem HTTPS que cause uma alteração de estado, a operação é ordenada de modo a ser executada por todas as réplicas corretas. Através da ordenação total providenciada pelo BFT-SMaRt e dado que todas as operações disponíveis são deterministas, garante-se que o estado das replicas se mantém consistente. Quando um servidor recebe um pedido de leitura, a operação não é ordenada; é apenas enviado o valor obtido por um quorum de leitura da operação nas réplicas. Assim o sistema assegura *Safety* e caso o sistema se encontre síncrono garante *Liveness*.

Cada servidor conhece a chave pública dos restantes. As chaves públicas dos servidores são utilizadas para que estes se autentiquem.

III. Modelo do Adversário

O sistema está protegido contra uma série de ataques por parte de um adversário. Assume-se que a superfície de ataque do adversário é composta pelas comunicações entre os clientes e os servidores, as comunicações entre os próprios servidores e possuir um conjunto limitado de servidores.

Assume-se que o adversário não consegue aceder aos dados de outro cliente e que, enquanto cliente, não realiza a operação *create-money* indevidamente. Esta operação cria uma quantidade arbitrária de dinheiro na conta de um utilizador, sendo esta operação também utilizada para a criação de utilizadores no sistema.

Sem comprometer o sistema, o adversário pode realizar *tampering*, *eavesdropping* e ataques de *replay* entre o cliente e um qualquer servidor. O sistema também está preparado para defender contra ataques de *mascarading* por parte do servidor. Deste modo o servidor está autenticado perante o cliente.

Nas comunicações entre os servidores, o adversário pode realizar ataques de *tampering*, *eavesdropping*, *replay*, *mascarading* e atrasar o envio de mensagens por tempo indefinido em no máximo, um terço dos servidores, sem que estes ataques comprometam o sistema.

Um adversário pode tomar controlo de um terço dos servidores, desde que nenhum destes servidores seja o ponto de contacto por parte de um cliente correto.

Assume-se que o adversário tem limitações em poder computacional, não conseguindo subverter técnicas criptográficas que utilizem chaves cuja dimensão é um padrão seguro actualmente.

IV. Modelo de falhas

O sistema suporta a existência de uma série de falhas nos seus constituintes, sendo que estas falhas podem resultar de um *crash* de componentes, ou de comportamento arbitrário por parte destes.

Assume-se que qualquer cliente está susceptível a falhas, desde que, caso um cliente seja apoderado por um adversário, este apenas realize as acções contempladas no modelo de adversário.

Assume-se que é possível que um terço dos servidores e correspondentes bases de dados possam falhar, sem comprometer o sistema.

V. Testes Realizados

Para realizar a auditoria do sistema foram realizados testes à tolerância a falhas, segurança e performance do sistema. Assim criaram-se quatro réplicas de servidores em *containers Docker*, cada um com uma base de dados MySQL associada, activos numa rede virtual isolada. Na rede privada cada réplica publica apenas uma porta na *NAT* do sistema de *containers* de forma a dar acesso apenas à *API Rest* através da qual são realizados as operações *HTTP* sobre *TLS*. Todas as operações entre réplicas e da base de dados são executadas no interior da rede privada.

VI. Segurança Comunicações

Como indicado no modelo do adversário apresentado na secção III, um atacante pode atacar as comunicações entre os servidores aplicacionais. Deste modo, é necessário que as comunicações entre estes sejam protegidas contra qualquer eventual ataque que figure no modelo do adversário.

Foi criada uma imagem Docker que executa o programa *tcpdump*. Posteriormente, criou-se um container com esta imagem associado a cada servidor. A utilização deste container permite que se consiga analisar o tráfego entre os vários servidores. O tráfego capturado foi posteriormente exportado para um ficheiro *pcap*, de forma a que pudessem ser analisados utilizando a ferramenta *Wireshark*. O ficheiro mencionado está presente na pasta ”Server/Auditoria”.

Entre os pacotes de dados transmitidos entre as replicas encontram-se os seguintes:

- Pacote 276 (Client Hello)
- Pacote 278 (Server Hello, Certificate,Server Key Exchange)
- Pacote 280 (Client Key Exchange)
- Pacote 282 (Change Cipher Suite)

Analisando os pacotes apresentados conclui-se que a comunicação entre as replicas fornecida pelo BFT-SMaRt é protegida pelo protocolo TLS, utilizando a seguinte *ciphersuite*:

- **Protocol**: Transport Layer Security (TLS 1.2)
- **Key Exchange**: Diffie-Hellman Ephemeral (DHE)
- **Authentication**: Digital Signature Standard (DSS)
- **Encryption**: Advanced Encryption Standard with 128bit key in Galois/Counter mode (AES 128 GCM)
- **Hash**: Secure Hash Algorithm 256 (SHA256)

A ciphersuite apresentada não contém características que induzam em falhas de segurança. Conclui-se assim que os dados que circulam entre os servidores do sistema se encontram protegidos contra adversários cujos recursos figurem no modelo de adversário apresentado.

VII. Tolerância a Falhas

Seja $\boldsymbol{f}$ o número máximo de servidores que podem falhar no sistema e $\boldsymbol{n}$ o número de servidores no total. Dado que o sistema suporta falhas Bizantinas, para que as operações sejam ordenadas é necessário que os servidores recebam um quorum com mais respostas de servidores correctos do que com falhas; deste modo um quorum deve ser constituído por, no mínimo, $2f + 1$ respostas.

Supondo que f servidores sofrem uma falha por *crash*, de modo a que seja possível constituir-se um quorum com a dimensão apresentada, é necessário que o número de servidores correctos seja superior a $2f + 1$. Assim, tem-se que $n - f \geq 2f + 1 \Leftrightarrow n \geq 3f + 1$. As asserções apresentadas são obtidas de [3].

Dado que se testou um sistema com 4 servidores, tem-se que no máximo uma falha é suportada, o que induz quorums de dimensão 3.

Testaram-se falhas de componentes resultantes de *crashes* e de comportamento Bizantino. Para testar as falhas correram-se as quatro réplicas em simultâneo e observou-se que o sistema tinha um comportamento normal. Posteriormente foram-se removendo réplicas, e, por cada réplica removida, testou-se se o sistema apresentava o comportamento esperado.

Simulando falhas por *crash* de componentes, comprovou-se que o sistema funciona correctamente perante uma falha, sendo os quorums constituídos por 3 respostas de servidores correctos. Quando existem duas falhas por *crash*, o número de respostas necessárias não é o suficiente para completar um quorum, pelo que o sistema fica bloqueado, esperando respostas e não realizando nenhuma operação.

Para a simulação de uma falha Bizantina, modificou-se o comportamento dos servidores, de modo a que um servidor Bizantino retornasse resultados incorrectos em operações de leitura. A ordenação das operações decorre correctamente, com ou sem falhas Bizantinas, contudo, as falhas são detectadas na avaliação dos resultados devolvidos. A simulação de falhas Bizantinas, envolve não só o comportamento do BFT-SMaRt, mas também a lógica de execução do sistema de leilões; para a realização de testes específicos ao comportamento do BT-SMaRt, as falhas Bizantinas devem ser causadas por erros no processo de ordenação das operações. Dado o comportamento do sistema de leilões ser correto, como averiguado na avaliação do WA1 e WA2; assume-se que qualquer inconsistência no comportamento esperado se deve à integração do BFT-SMaRt.

Simulando falhas Bizantinas, tem-se novamente que o sistema funciona correctamente na presença de uma falha. Na circunstância apresentada, os quorums recebidos para a ordenação das operações são constituídos por três respostas corretas, ou duas respostas correctas e uma incorrecta. Independentemente do número de respostas correctas, estas encontram-se sempre em maioria. Na presença de duas falhas bizantinas, o sistema fica bloqueado, indicando que não é possível obter-se um quorum de dimensão mínima com respostas com o mesmo valor.

Foi executado um teste no qual se introduziu uma réplica Bizantina e simulou-se o *crash* de outra. Novamente, o sistema bloqueou esperando um quorum de respostas com o mesmo valor.

VIII. Avaliação de Performance

```
duration: -, iterations: 10000
     vus: 1, max: 1

init [------------------------------------------------------------] starting
data_received..............: 1.4 MB 19 kB/s
data_sent..................: 990 kB 14 kB/s
http_req_blocked...........: avg=2ms      min=160µs  med=318µs  max=14.5s  p(90)=612µs   p(95)=778.09µs
http_req_connecting........: avg=1.92ms   min=119µs  med=240µs  max=14.5s  p(90)=517µs   p(95)=663.04µs
http_req_duration..........: avg=5.17ms   min=1.51ms med=4.01ms max=1.15s  p(90)=6.26ms  p(95)=7.32ms
http_req_receiving.........: avg=351.17µs min=31µs   med=87µs   max=1s     p(90)=583µs   p(95)=842.04µs
http_req_sending...........: avg=70.48µs  min=24µs   med=59µs   max=1.07ms p(90)=112.1µs p(95)=138µs
http_req_tls_handshaking...: avg=0s       min=0s     med=0s     max=0s     p(90)=0s      p(95)=0s
http_req_waiting...........: avg=4.75ms   min=1.45ms med=3.73ms max=1.15s  p(90)=5.83ms  p(95)=6.86ms
http_reqs..................: 10000  137.345527/s
iteration_duration.........: avg=7.25ms   min=1.73ms med=4.45ms max=14.5s  p(90)=6.9ms   p(95)=7.91ms
iterations.................: 10000  137.345527/s
vus........................: 1      min=1 max=1
vus_max....................: 1      min=1 max=1
```

```
duration: -, iterations: 10000
     vus: 1, max: 1

init [------------------------------------------------------------] starting
data_received..............: 1.4 MB 19 kB/s
data_sent..................: 990 kB 14 kB/s
http_req_blocked...........: avg=1.97ms   min=159µs  med=318µs  max=14.5s   p(90)=613µs   p(95)=782µs
http_req_connecting........: avg=1.89ms   min=112µs  med=240µs  max=14.5s   p(90)=513µs   p(95)=669.04µs
http_req_duration..........: avg=5.21ms   min=1.8ms  med=4.03ms max=1.15s   p(90)=6.27ms  p(95)=7.43ms
http_req_receiving.........: avg=234.86µs min=34µs   med=88µs   max=27.85ms p(90)=578.1µs p(95)=842.04µs
http_req_sending...........: avg=70.82µs  min=26µs   med=59µs   max=597µs   p(90)=114µs   p(95)=143µs
http_req_tls_handshaking...: avg=0s       min=0s     med=0s     max=0s      p(90)=0s      p(95)=0s
http_req_waiting...........: avg=4.91ms   min=1.63ms med=3.75ms max=1.15s   p(90)=5.84ms  p(95)=6.9ms
http_reqs..................: 10000  137.034999/s
iteration_duration.........: avg=7.27ms   min=1.98ms med=4.48ms max=14.5s   p(90)=6.84ms  p(95)=8.02ms
iterations.................: 10000  137.034999/s
vus........................: 1      min=1 max=1
vus_max....................: 1      min=1 max=1
```

```
duration: -, iterations: 10000
     vus: 1, max: 1

init [------------------------------------------------------------] starting
data_received..............: 1.4 MB 19 kB/s
data_sent..................: 990 kB 14 kB/s
http_req_blocked...........: avg=1.97ms   min=135µs  med=317µs  max=14.5s  p(90)=608.1µs p(95)=778.04µs
http_req_connecting........: avg=1.89ms   min=106µs  med=240µs  max=14.5s  p(90)=512µs   p(95)=661µs
http_req_duration..........: avg=5.21ms   min=1.87ms med=4.02ms max=1.15s  p(90)=6.26ms  p(95)=7.35ms
http_req_receiving.........: avg=232.58µs min=31µs   med=87µs   max=9.48ms p(90)=576.1µs p(95)=850µs
http_req_sending...........: avg=70.65µs  min=27µs   med=59µs   max=931µs  p(90)=114µs   p(95)=141µs
http_req_tls_handshaking...: avg=0s       min=0s     med=0s     max=0s     p(90)=0s      p(95)=0s
http_req_waiting...........: avg=4.9ms    min=1.79ms med=3.74ms max=1.15s  p(90)=5.83ms  p(95)=6.85ms
http_reqs..................: 10000  137.162822/s
iteration_duration.........: avg=7.26ms   min=2.13ms med=4.48ms max=14.51s p(90)=6.87ms  p(95)=7.97ms
iterations.................: 10000  137.162822/s
vus........................: 1      min=1 max=1
vus_max....................: 1      min=1 max=1
```

Fig. 2. Tempos execução de operações *read* utilizando BFT-SMaRt

A utilização de mecanismos de state machine replication incorre perdas de performance derivadas da latência da comunicação entre as replicas na agregação dos quorums para leituras e ordenação de operações. Deste modo justifica-se a realização de testes de performance para que sejam realizadas medições de tempo com e sem o BFT-SMaRt.

Para testar a performance do sistema, foi utilizando o *benchmark K6*. Neste teste, as várias operações que o sistema disponibiliza foram testadas, sendo o tempo da realização destas medido e analisado. Os testes foram realizados utilizando uma máquina com um processador *2,6 GHz Intel Core i7* de 6 CPU e memória de *6 GB 2400 MHz DDR4*.

```
duration: -, iterations: 10000
     vus: 1, max: 1

init [------------------------------------------------------------] starting
data_received..............: 3.0 MB 429 kB/s
data_sent..................: 990 kB 140 kB/s
http_req_blocked...........: avg=134.82µs min=78µs     med=99µs     max=3.56ms  p(90)=220.1µs  p(95)=363µs
http_req_connecting........: avg=107.94µs min=58µs     med=74µs     max=3.1ms   p(90)=192µs    p(95)=337µs
http_req_duration..........: avg=540.2µs  min=446µs    med=504µs    max=12.54ms p(90)=614µs    p(95)=680µs
http_req_receiving.........: avg=99.58µs  min=23µs     med=91µs     max=12.11ms p(90)=116µs    p(95)=132µs
http_req_sending...........: avg=24.54µs  min=14µs     med=24µs     max=732µs   p(90)=27µs     p(95)=31µs
http_req_tls_handshaking...: avg=0s       min=0s       med=0s       max=0s      p(90)=0s       p(95)=0s
http_req_waiting...........: avg=416.06µs min=351µs    med=389µs    max=12.22ms p(90)=474µs    p(95)=531µs
http_reqs..................: 10000  1414.424068/s
iteration_duration.........: avg=700.02µs min=557.45µs med=630.53µs max=12.66ms p(90)=860.59µs p(95)=991.1µs
iterations.................: 10000  1414.424068/s
vus........................: 1      min=1 max=1
vus_max....................: 1      min=1 max=1
```

Fig. 3. Tempos de execução de operações *read* sem a utilização de BFT-SMaRt

Dado que Na realização de operações de leitura, o BFT-SMaRt requer um quorum de $2f + 1$ respostas. No sistema testado, traduz-se em quorums de três respostas.

A figura 2 apresenta o output do *benchmark* correndo-se três processos em paralelo, com cada processo realizando 10000 operações de leitura.

Analisando os valores do *benchmark* tem-se que o tempo médio de duração de cada leitura são *5.17ms*, *5.21ms* e *5.21ms* respectivamente. Assumindo que a distribuição de tempos entre os três processos são independentes e identicamente distribuídos, estima-se que o tempo médio de resposta a uma leitura, perante pedidos concorrentes seja de *5.2ms*.

A figura 3 representa o output do *benchmark* no qual se realizaram 10000 operações de leitura no sistema de leilões sem que a integração do BFT-SMaRt. Analisando os resultados tem-se o tempo médio de leituras é de $540.2\mu s$. Deste modo comprova-se que a utilização do BFT-SMaRt produz leituras *963%* mais lentas.

As operações de escrita são ordenadas pelo BFT-SMaRt, de modo a garantir a consistência forte do sistema, bem como tolerância a falhas bizantinas. O processo executado pelo BFT-SMaRt requer que sejam realizados três quorums com $2f + 1$ respostas.

A figura 4 apresenta o output do *benchmark* obtido pela execução de três processos em paralelo, sendo que cada processo realizou 10000 operações de escrita. Cada processo envia as operações a realizar a um servidor diferente dos restantes, simulando assim o comportamento esperado na execução normal do sistema de leilões por um vasto conjunto de clientes.

Analisando os resultados obtidos tem-se que os tempos médios das operações de escrita para cada processo são *5.78ms*, *5.82ms* e *5.64ms*. Assumindo que as medições de tempo entre as três réplicas são i.i.d, estima-se que o tempo médio de resposta a uma operação de escrita seja de *5.74ms*.

O output do *benchmark* quando não se utiliza BFT-SMaRt está apresentado na figura 5; os resultados obtiveram-se realizando 10000 operações de escrita através de um único processo. Como se pode comprovar pelos resultados, o tempo médio de cada operação de escrita é $579\mu s$. Deste modo tem-se que a utilização do BFT-SMaRt incorre numa perda de performance de 991% em operações de escrita.

O processo de replicação de dados com consistência forte é a operação mais complexa do sistema, por este motivo, a

```
duration: -, iterations: 10000
     vus: 1, max: 1

init [------------------------------------------------------------] starting
data_received..............: 1.4 MB 18 kB/s
data_sent..................: 1.7 MB 23 kB/s
http_req_blocked...........: avg=1.65ms   min=160µs  med=308µs  max=12.39s  p(90)=598µs  p(95)=811µs
http_req_connecting........: avg=1.56ms   min=113µs  med=232µs  max=12.39s  p(90)=486µs  p(95)=691µs
http_req_duration..........: avg=5.78ms   min=2.08ms med=4.46ms max=4.85s   p(90)=7.27ms p(95)=8.57ms
http_req_receiving.........: avg=290.58µs min=35µs   med=93µs   max=49.43ms p(90)=718µs  p(95)=1.02ms
http_req_sending...........: avg=78.47µs  min=31µs   med=65µs   max=1.09ms  p(90)=124µs  p(95)=160µs
http_req_tls_handshaking...: avg=0s       min=0s     med=0s     max=0s      p(90)=0s     p(95)=0s
http_req_waiting...........: avg=5.41ms   min=1.92ms med=4.12ms max=4.85s   p(90)=6.75ms p(95)=7.99ms
http_reqs..................: 10000  131.283391/s
iteration_duration.........: avg=7.59ms   min=2.44ms med=4.96ms max=12.4s   p(90)=7.94ms p(95)=9.28ms
iterations.................: 10000  131.283391/s
vus........................: 1      min=1 max=1
vus_max....................: 1      min=1 max=1
```

```
duration: -, iterations: 10000
     vus: 1, max: 1

init [------------------------------------------------------------] starting
data_received..............: 1.4 MB 18 kB/s
data_sent..................: 1.7 MB 23 kB/s
http_req_blocked...........: avg=1.61ms   min=163µs  med=308µs  max=12.37s  p(90)=609µs   p(95)=821µs
http_req_connecting........: avg=1.53ms   min=119µs  med=232µs  max=12.37s  p(90)=499µs   p(95)=691.14µs
http_req_duration..........: avg=5.82ms   min=2.04ms med=4.46ms max=4.85s   p(90)=7.24ms  p(95)=8.75ms
http_req_receiving.........: avg=288.86µs min=29µs   med=93µs   max=34.43ms p(90)=715µs   p(95)=1.05ms
http_req_sending...........: avg=78.16µs  min=31µs   med=65µs   max=772µs   p(90)=123.1µs p(95)=159µs
http_req_tls_handshaking...: avg=0s       min=0s     med=0s     max=0s      p(90)=0s      p(95)=0s
http_req_waiting...........: avg=5.45ms   min=1.9ms  med=4.12ms max=4.85s   p(90)=6.75ms  p(95)=8.14ms
http_reqs..................: 10000  131.235202/s
iteration_duration.........: avg=7.59ms   min=2.38ms med=4.98ms max=12.38s  p(90)=7.91ms  p(95)=9.45ms
iterations.................: 10000  131.235202/s
vus........................: 1      min=1 max=1
vus_max....................: 1      min=1 max=1
```

```
duration: -, iterations: 10000
     vus: 1, max: 1

init [------------------------------------------------------------] starting
data_received..............: 1.4 MB 18 kB/s
data_sent..................: 1.7 MB 23 kB/s
http_req_blocked...........: avg=1.72ms   min=150µs  med=310µs  max=12.35s p(90)=584µs   p(95)=779.09µs
http_req_connecting........: avg=1.64ms   min=110µs  med=233µs  max=12.35s p(90)=474µs   p(95)=652.04µs
http_req_duration..........: avg=5.64ms   min=2.09ms med=4.41ms max=4.86s  p(90)=7.1ms   p(95)=8.52ms
http_req_receiving.........: avg=796.39µs min=33µs   med=103µs  max=4.85s  p(90)=757µs   p(95)=1.07ms
http_req_sending...........: avg=78.55µs  min=31µs   med=65µs   max=1.09ms p(90)=123µs   p(95)=158µs
http_req_tls_handshaking...: avg=0s       min=0s     med=0s     max=0s     p(90)=0s      p(95)=0s
http_req_waiting...........: avg=4.76ms   min=1.82ms med=4.06ms max=1.19s  p(90)=6.57ms p(95)=7.93ms
http_reqs..................: 10000  132.388843/s
iteration_duration.........: avg=7.52ms   min=2.39ms med=4.93ms max=12.36s p(90)=7.82ms p(95)=9.24ms
iterations.................: 10000  132.388843/s
vus........................: 1      min=1 max=1
vus_max....................: 1      min=1 max=1
```

Fig. 4. Tempos de execução de operações *write* utilizando BFT-SMaRt

```
duration: -, iterations: 10000
     vus: 1, max: 1

init [------------------------------------------------------------] starting
data_received..............: 732 kB 114 kB/s
data_sent..................: 1.7 MB 268 kB/s
http_req_blocked...........: avg=3.12µs   min=0s       med=1µs      max=1.1ms   p(90)=2µs      p(95)=2µs
http_req_connecting........: avg=1.29µs   min=0s       med=0s       max=229µs   p(90)=0s       p(95)=0s
http_req_duration..........: avg=578.47µs min=437µs    med=495µs    max=94.17ms p(90)=753µs    p(95)=893µs
http_req_receiving.........: avg=13.87µs  min=7µs      med=12µs     max=2.02ms  p(90)=19µs     p(95)=25µs
http_req_sending...........: avg=9.56µs   min=4µs      med=9µs      max=264µs   p(90)=13µs     p(95)=16µs
http_req_tls_handshaking...: avg=0s       min=0s       med=0s       max=0s      p(90)=0s       p(95)=0s
http_req_waiting...........: avg=555.03µs min=423µs    med=474µs    max=94.05ms p(90)=718.1µs  p(95)=853µs
http_reqs..................: 10000  1559.801315/s
iteration_duration.........: avg=635.02µs min=476.12µs med=544.79µs max=95.45ms p(90)=829.84µs p(95)=987.84µs
iterations.................: 10000  1559.801315/s
vus........................: 1      min=1 max=1
vus_max....................: 1      min=1 max=1
```

Fig. 5. Tempos de execução de operações *write* sem a utilização de BFT-SMaRt

perda de performance incorrida pela utilização do BFT-SMaRt não deve interpretada como evitável em sistemas replicados. Para uma percepção mais informativa dos tempos recolhidos, deve-se comparar estes com aqueles que seriam recolhidos integrando outro sistema de replicação de máquinas de estado no sistema de leilões.

IX. Conclusão

Na auditoria realizada foram apresentados e testados vários aspectos do sistema. Como descrito em [1, 2] as comunicações entre as replicas são protegidas pelo protocolo TLS. Tomando

como ponto de vista um adversário cujos recursos se encontrem descritos no modelo do adversário, tem-se que não é possível inferir informações confidenciais dos dados em transito, nem provocar falhas no sistema através de mensagens corrompidas.

Pelos testes realizados referentes à tolerância a falhas tem-se que estes seguem o modelo proposto originalmente por [3]. Deste modo conclui-se que o BFT-SMaRt foi corretamente integrado no sistema de leilões implementado.

Através da análise de performance tem-se que a utilização do BFT-SMaRt permite que o sistema tenha um sistema de replicação forte com tolerância a falhas Bizantinas sem incorrer em perdas de performance que inviabilizem a utilização do sistema.

A prioridade da implementação do sistema de leilões não é a segurança do sistema, pelo que se escusa o fraco modelo de adversário proposto. Num sistema mais robusto, ter-se-ia autenticação por parte do cliente, um sistema de controlo de acessos integrado e a exclusão da possibilidade dos clientes realizarem a operação *create-money*. Outros mecanismos a nível aplicacional poderiam ainda ser desenvolvidos para a protecção contra ataques de *DoS*.

Uma falha de segurança grave que o sistema apresentado não contempla é a situação em que um cliente contacta um servidor Bizantino. Um servidor Bizantino pode recusar realizar as operações de um cliente, ou enviar a resposta errada; na implementação actual, o cliente não tem meios para detectar ou proteger-se das acções do servidor Bizantino. Outra falha possível é um servidor Bizantino iniciar operações por parte de um cliente, sem que o cliente as tenha mandado executar.

Uma solução para ambos os problemas é ter-se um cliente mais complexo que receba respostas de todas as réplicas e verifique qual a resposta correta escolhida com base no quão frequente é a resposta entre as obtidas. Segundo [1, 2], utilizando o BFT-SMaRt, uma solução para o problema apresentado obtêm-se tendo o cliente contactando todas as replicas por cada operação que deseje realizar. A solução apresentada é adequada para operações de leitura, contudo, se utilizada em operações de escrita resulta que a operação é realizada várias vezes, produzindo resultados diferentes entre as replicas correctas e alterando o estado do sistema contra as intenções do utilizador.

Uma solução alternativa encontra-se em [4] no qual o cliente apenas contacta uma replica, mas recebe uma resposta de todas as replicas.

O foco na modularidade por parte do BFT-SMaRt [1, 2] permite que este seja integrado facilmente em um sistema distribuído, requerendo poucas, ou nenhumas, alterações na estrutura do sistema. Apesar da integração com sistema de leilões apresentar falhas evitáveis, o projecto é um testemunho à versatilidade do BFT-SMaRt.

REFERENCES

[1] A. Bessani, J. Sousa, E. Alchieri. State Machine Replication for the masses with BFT-SMaRt

[2] J. Sousa and A. Bessani, "From Byzantine consensus to BFT state machine replication: A latency-optimal transformation," em Proc. of EDCC'12, 2012.

[3] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, 4(3), 1982.

[4] M. Castro and B. Liskov. Practical Byzantine fault-tolerance and proactive recovery. ACM Transactions Computer Systems, 20(4):398–461, 2002.