TÉCNICO LISBOA

Dependable Public Announcement System

SEC
Grupo 18

Carolina Carreira 87641

Catarina Pedreira 87524

Miguel Barros 87691

TÉCNICO LISBOA

1. Explicação do Design

Para poder tolerar eventuais faltas de servidores, o sistema foi replicado por **N** réplicas, em que **N** depende do número de faltas (**f**) que queremos tolerar (**N = 3 * f + 1).** Este valor foi escolhido porque garante que existe sempre um quórum bizantino na presença de **f** faltas (de tamanho 2 * f + 1).

Esta replicação exige um sistema de coordenação, para garantir que todos os clientes do sistema veem um estado consistente. Deste modo, os boards dos utilizadores são modelados como **(1,N) Byzantine Atomic Register** e o board geral como um **(N, N) Byzantine Regular Register**.

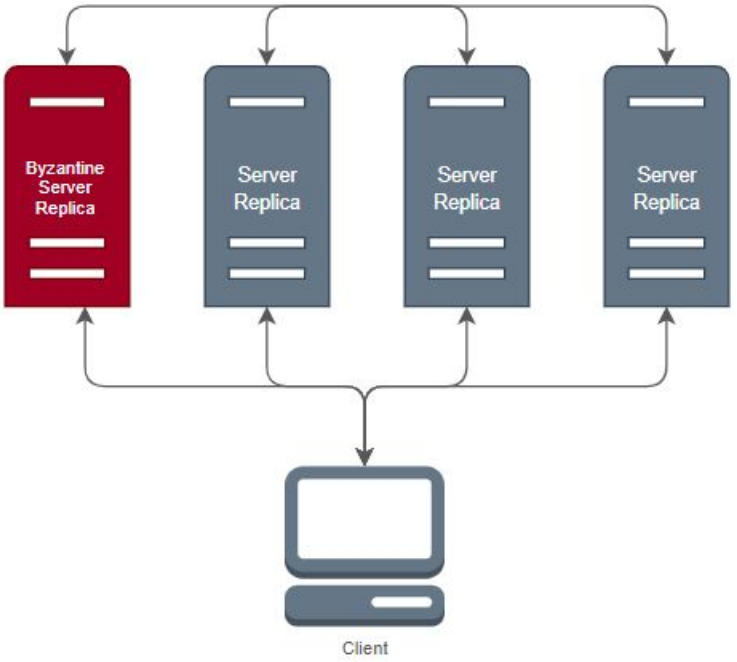

Figura 1: Funcionamento de sistema com um servidor bizantino e um cliente correcto

A especificação foi alterada para que uma escrita no board seja um append de um post nesse board. Todos os posts têm associados um timestamp lógico.

Os algoritmos dos registos requerem **Authenticated Perfect Links (APL)** que garantam a entrega das mensagens aos servidores. A nossa implementação consiste em repetir uma mensagem até obtermos um **ACK** devidamente autenticado da parte do servidor. Também aplicamos isto à comunicação entre servidores.

Como assumimos que o cliente que usa a nossa biblioteca sem alterações está correto, se ocorrer uma excepção no lado do servidor só poderá ter sido um servidor bizantino ou um atacante que de alguma maneira causou a excepção (alterou a mensagem enviada ou a resposta do servidor).

Para garantir que operações não são repetidas do lado do servidor, a sua unicidade é verificada pelo seu **MAC**. Caso receba uma mensagem com um *MAC* antigo, retorna simplesmente um ACK mas não efetua a operação novamente (excepto no caso dos reads, em que um nonce do cliente garante que o MAC é sempre único, caso o servidor já tenha visto esse MAC retorna uma excepção).

User Board (1,N) Byzantine Atomic Register:

Read: Consiste em enviar um *ReadRequest* a todos os servidores e esperar por um quórum bizantino de respostas. Cada resposta é autenticada (pelo *APL*). Das respostas obtidas, escolhemos a mais recente através do seu *timestamp*. Pelas propriedades dos quóruns, veremos a escrita mais recente em pelo menos um servidor correto.

Para garantir a propriedade de *linearizability* do registo, é adicionada ao read uma fase de *write-back* em que o cliente efetua *Write* do announcement mais recente que observou.

Write: Consiste em enviar um *Announcement* a todos aos servidores e esperar por um quórum bizantino de ACKs. O *announcement* enviado tem o timestamp *t+1* em que t e o maior timestamp dos announcements desse board. Se o cliente não souber t efetua um read para o obter.

General Board (N,N) Byzantine Regular Register:

Read: Similar ao anterior, mas eliminando a fase de *write-back*. Como podemos ter múltiplos escritores concorrentes, podemos ter posts com o mesmo *timestamp*. Estes são desempatados pela chave pública do utilizador.

Write: Similar ao anterior, mas adicionando uma fase de read no início para obter o *timestamp* atual do board, que é depois incrementado e passa a ser o *timestamp* do novo announcement.

Assunções

- Assumimos uma distribuição de chaves em que os clientes têm as chaves dos servidores e de outros clientes *a priori*.
- Assumimos que um atacante não altera a biblioteca de um cliente correto (ou seja, a biblioteca é *trusted*).

2. Propriedades

Autenticação, Não Repúdio e Integridade

Através dos *APL* e assinaturas. Todos os *Posts* são assinados pelo user, que os cria com a sua chave privada. Estas propriedades são respeitadas por todas as operações. O *Register* e o *Read* também contêm MACs que são inforjáveis.

Unicidade

Unicidade é obtida através de Sequence Numbers dos posts. Se o servidor observar um sequence number que não espera (por exemplo, só conhece o announcement com *timestamp* 0 e observa um com *timestamp* 10) envia uma excepção. Caso o servidor esteja desatualizado ele irá eventualmente receber os *announcements* em falta (pelo *APL*). Isto impede um cliente bizantino de exaustar o espaço dos *announcements*.

TÉCNICO LISBOA

Persistência

O estado do sistema é persistente, e este sistema é guardado de forma atómica. No entanto, um servidor que *crash* conta para as f faltas toleradas, podendo no entanto ser recuperado e continuar a participar no sistema.

Disponibilidade

Devido ao facto de os servidores estarem replicados e apenas ser necessário um quórum para fazer decisões, este consegue funcionar na presença de f faltas.

Concorrência

Foram efetuados testes com clientes concorrentes e servidores replicados e validado o estado final do sistema.

Comunicação

Devido ao uso de *APL* e *Byzantine Reliable Broadcast,* as seguintes propriedades são garantidas para cada mensagem m:

- **Validity**: Se um processo correto p faz broadcast de uma mensagem m, todos os processos corretos vão eventualmente entregar m.
- **No creation:** Nenhuma mensagem é entregue a não ser que tenha sido enviada.
- **No duplication:** Nenhuma mensagem é entregue mais do que uma vez num processo.
- **Integrity:** Se um processo correto entrega uma mensagem m com um sender p, e p é correto, então a mensagem m foi previamente broadcasted por p.
- **Consistency:** Se um processo correto entrega uma mensagem m e outro processo correto entrega uma mensagem m', então m = m'.

3. Análise de ameaças e mecanismos de proteção

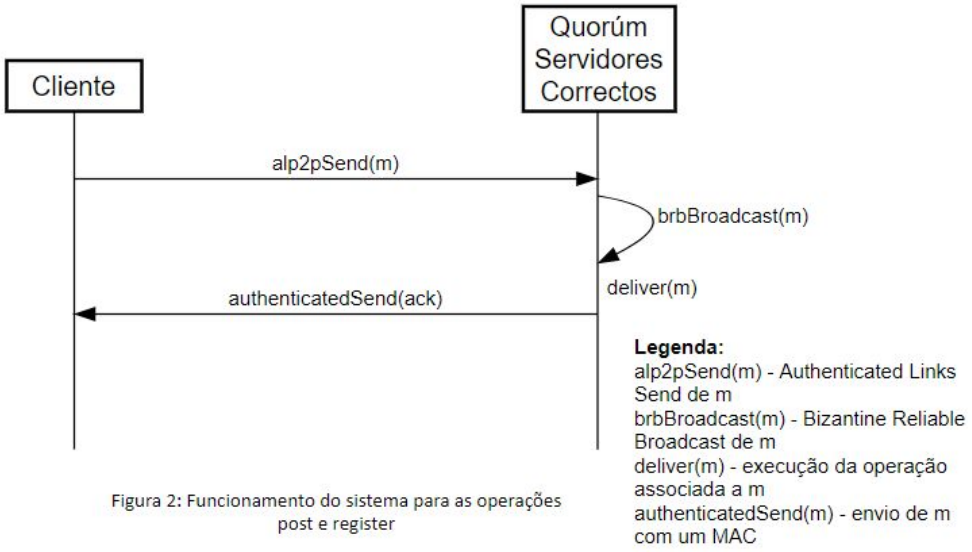

Figura 2: Funcionamento do sistema para as operações post e register

- **Byzantine Clients**

Protegemos o nosso sistema desta ameaça através da execução de um *Byzantine Reliable Broadcast* (algoritmo *Authenticated Double Echo Broadcast*) para as operações *Register* e *Post* antes de as entregar a mensagem à aplicação do servidor. Isto garante a propriedade de consistency. Um Post enviado a um só servidor não é entregue à aplicação.

Se um cliente bizantino e um servidor bizantino tentassem cooperar para danificar o sistema, poderiam por exemplo efetuar um post que bloqueia outros clientes de progredirem por ter um *timestamp* demasiado elevado. Para evitar isto cada servidor assina cada mensagem “ready” que envia (consequência do uso de *APL*), e coleciona todas as mensagens “ready” que recebeu. Posteriormente envia-as para o cliente na operação *Read* para que este possa verificar que ocorreu um *broadcast* dos *Posts* retornados.

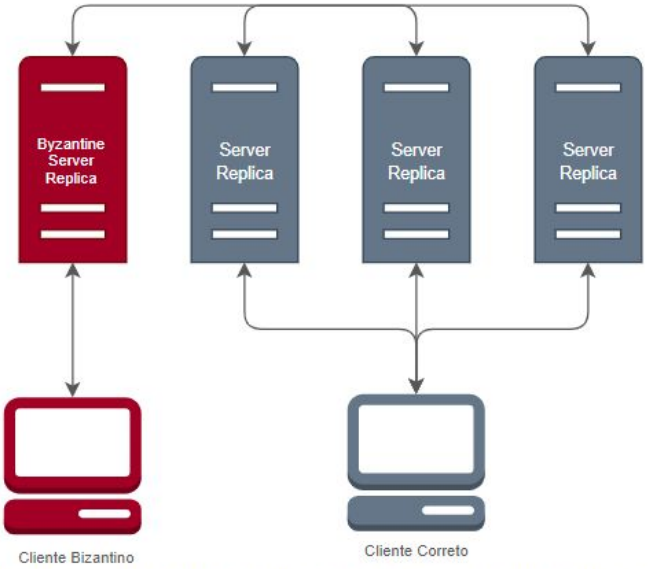

Figura 3: Funcionamento de sistema com um servidor bizantino e um cliente bizantino a cooperarem.

- **MITM (Man-in-the-middle)**

  O nosso sistema está protegido desta ameaça através das propriedades de Autenticação, Não Repúdio e Integridade dos *APL*.

- **Replay attacks**

  Impossível pela propriedade *No Duplication* dos *APL*.