Um Estudo sobre o Uso de Replicação Máquina de Estados Paralelas na Implementação de Blockchains

Aldênio Burgos[1] **e Eduardo Alchieri**[1]

[1]Departamento de Ciência da Computação
Universidade de Brasília
Brasília - Brasil

Abstract. *State Machine Replication (SMR) and blockchains share the same goal of keep a consistent state replicated across a set of replicas. However, there are some subtle differences between these techniques, as the way logs are created and used. Moreover, using a SMR framework as a building block to implement blockchains significantly impacts performance due to the sequential execution model of SMRs. This work presents a case study that uses a parallel SMR in the implementation of blockchains. Through the implementation of a payment system and a set of experiments, this work shows that, by using parallel SMRs, it is possible to circumvent the previously described performance limitation, i.e., the system performance increases substantially.*

Resumo. *Replicação Máquina de Estados (RME) e blockchains possuem um objetivo em comum que é o de manter a consistência no estado de um serviço replicado. Porém, algumas diferenças fundamentais são encontradas entre estes modelos, como por exemplo a forma de manutenção e utilização do log de requisições. Além disso, a utilização de uma RME como um bloco de construção para blockchains impacta significativamente o desempenho do sistema devido principalmente ao modelo de execução sequencial das RMEs. Neste trabalho, apresentamos um estudo de caso que utiliza uma solução para RME que possibilita a execução paralela de uma parte das requisições/transações na implementação de blockchains. Através da implementação de um sistema de pagamento e a realização de uma série de experimentos, este trabalho mostra que, usando RMEs paralelas, a limitação anteriormente descrita pode ser minimizada, i.e., o desempenho do sistema é aumentado substancialmente.*

1. Introdução

A Replicação Máquina de Estados (RME) [Schneider 1990] é uma abordagem muito utilizada na implementação de sistemas replicados. Esta abordagem é usada para manter a consistência do estado de um conjunto de réplicas, as quais executam deterministicamente o mesmo conjunto de requisições na mesma ordem. RMEs tolerantes a falhas bizantinas (BFT) [Castro and Liskov 1999, Castro and Liskov 2002] são particularmente relevantes no contexto de *blockchains* permissionadas [Cachin and Vukolic 2017], por suportarem que uma fração das réplicas sejam controladas por um adversário. Estes protocolos também são usados em *blockchains* não permissionadas, onde um conjunto de réplicas é escolhido para formar um comitê que executa os protocolos de replicação (exemplo, [Pass and Shi 2017, Abraham et al. 2017]).

Apesar de protocolos para RMEs tolerantes a falhas bizantinas e para *blockchains* compartilharem o objetivo de manter o estado replicado e consistente entre as várias réplicas, existem importantes diferenças entre estas duas abordagens [Bessani et al. 2020]. Primeiramente, enquanto que as RMEs mantêm um *log* de requisições executadas para sincronização de estados, por exemplo após a recuperação de uma réplica, nas *blockchains* este *log* precisa (1) ser escrito em memória estável (disco) para garantir durabilidade, (2) incluir o resultado da execução das requisições/transações para permitir auditoria, e (3) precisa ser auto-verificável por qualquer outra parte. Além disso, outra diferença é que em *blockchains*, ao contrário de RMEs, o conjunto de réplicas geralmente sofre reconfigurações sem o auxílio de partes confiáveis. Finalmente, um trabalho recente [Bessani et al. 2020] demonstrou que utilizar RMEs como um bloco de construção na implementação de *blockchains* resulta em um sistema com limitação de desempenho, principalmente devido a execução sequencial das transações, que envolvem custosas operações criptográficas (i.e., verificação de assinaturas digitais).

Neste contexto, este trabalho busca estudar como uma abordagem paralela para RMEs poderia melhorar o desempenho de *blockchains*, minimizando a limitação anteriormente discutida. RMEs paralelas tiram proveito da semântica das requisições para viabilizar a execução paralela de um subconjunto delas. Estas abordagens classificam as requisições em dependentes (ou conflitantes) e independentes (ou não conflitantes), de modo que as requisições independentes são executadas em paralelo enquanto que as requisições dependentes devem ser executadas sequencialmente. Duas requisições são independentes quando acessam diferentes variáveis ou apenas leem o valor de uma mesma variável. Por outro lado, duas requisições são dependentes quando acessam pelo menos uma variável em comum e ao menos uma das requisições altera o valor desta variável. Note que muitos trabalhos recentes focaram na escalabilidade do protocolo de consenso subjacente (exemplo, [Golan-Gueta et al. 2019], [Liu et al. 2019] e [Guerraoui et al. 2019]), mas no presente estudo estamos particularmente interessados na camada de execução das transações.

Para o presente estudo, utilizamos um sistema de pagamento instantâneo implementado através de moedas digitais que suporta transações para a transferência de valores entre contas de usuários, consulta de saldos, câmbio entre diferentes moedas, dentre outras. Os resultados experimentais mostram que o desempenho do sistema é substancialmente maior através da execução paralela de algumas destas operações (por exemplo, aquelas envolvendo contas distintas), i.e., através de uso de uma RME paralela.

O restante deste texto é organizado da seguinte forma. A Seção 2 apresenta o modelo de sistema, além de discutir RMEs paralelas e sistemas de pagamentos. A Seção 3 apresenta a moeda digital utilizada neste estudo, enquanto que a Seção 4 discute os experimentos realizados. Finalmente, a Seção 5 conclui o trabalho.

2. Definições Preliminares

Esta seção detalha o modelo de sistema adotado, a solução para RME paralela utilizada neste estudo, além de *blockchains* e sistemas de pagamentos.

2.1. Modelo de Sistema

Assumimos um sistema distribuído composto de processos interconectados. Há um conjunto potencialmente ilimitado de processos cliente e um conjunto de n processos servidor

(réplicas). O sistema é assíncrono, i.e., não há limite para atrasos de mensagens e velocidades relativas de processo. Assumimos o modelo de falha bizantina ou maliciosa. Um processo é *correto* se não falha, ou *faltoso* caso contrário. Existem no máximo f réplicas faltosas, de $n = 3f + 1$ réplicas.

Os processos se comunicam por envio de mensagens, usando comunicação ponto-a-ponto ou difusão atômica. A comunicação ponto-a-ponto usa as primitivas $send(m)$ e $receive(m)$, onde m é uma mensagem. Se um remetente enviar uma mensagem vezes suficientes, um destinatário correto eventualmente receberá a mensagem. A difusão atômica é definida pelas primitivas $broadcast(m)$ e $deliver(m)$, onde m é uma mensagem, e garante as seguintes propriedades [Défago et al. 2004, Hadzilacos and Toueg 1993] [1]:

- *Validade*: Se um processo correto difunde a mensagem m, então algum processo correto eventualmente entrega m.
- *Acordo uniforme*: Se um processo correto entrega a mensagem m, então todos os processos corretos eventualmente entregarão m.
- *Integridade uniforme*: Para qualquer mensagem m, se m foi transmitida por um processo, todo processo correto entregará m uma única vez.
- *Ordem total uniforme*: Se dois processos p e q entregam as mensagens m e m', então p entrega m antes de m', se e somente se, q entrega m antes de m'.

Protocolos de RME proveem consistência forte ou *linearizabilidade* [Herlihy and Wing 1990]. Uma execução é linearizável se houver uma maneira de ordenar totalmente as operações de tal forma que (a) respeite a semântica dos objetos acessados pelas operações, expressa em suas especificações sequenciais; e (b) respeite a ordenação no tempo das operações. Existe uma ordem no tempo entre duas operações se uma operação termina em um cliente antes que a outra operação comece em outro cliente.

2.2. Replicação Máquina de Estados Paralela

Para melhorar o desempenho do sistema, RMEs paralelas geralmente utilizam escalonadores que tiram proveito da semântica das requisições e permitem a execução paralela de algumas delas. Estas abordagens exploram a concorrência entre requisições. Assumindo que R é o conjunto de requisições possíveis, então $\#_R \subseteq R \times R$ é a relação de conflito entre as requisições. Se $\{r_i, r_j\} \in \#_R$, então as requisições r_i e r_j conflitam e suas execuções devem ser serializadas; caso contrário, r_i e r_j podem ser executadas simultaneamente. De uma forma geral, a ideia básica de RMEs paralelas é coordenar um conjunto de *threads* para executar em paralelo requisições que não conflitam e sequencialmente as requisições que conflitam entre si.

Os escalonadores para RMEs paralelas podem ser classificados em três categorias. No *escalonamento tardio* [Kotla and Dahlin 2004, Escobar et al. 2019] as requisições são escalonadas para execução após serem ordenadas pelas réplicas, enquanto que no *escalonamento antecipado* [Alchieri et al. 2017, Alchieri et al. 2018] parte das decisões de escalonamento são realizadas antes da ordenação e precisam ser respeitadas durante a execução. Por fim, o *escalonamento híbrido* [Burgos et al. 2021] busca aproveitar as vantagens de cada uma das soluções anteriores, sendo uma combinação de ambas as técnicas anteriores.

[1] Difusão atômica exige suposições adicionais de sincronia para implementação [Fischer et al. 1985].

Para este estudo, utilizaremos o escalonamento híbrido por apresentar o melhor desempenho dentre os escalonadores existentes. De forma resumida, neste tipo de escalonamento o estado da aplicação é particionado e os clientes devem identificar as partições (moedas) acessadas por suas requisições. As réplicas ordenam as requisições usualmente, como em uma RME tradicional. Porém, após a entrega, uma *thread*, chamada de classificador, encaminha as requisições uma-a-uma para diferentes *threads* paralelizadoras. Em cada réplica, as *threads* paralelizadoras inserem as requisições paralelamente em um grafo de conflitos. Um subgrafo é associado a cada partição e um paralelizador fica responsável por cada subgrafo. Para cada requisição endereçada a mais de uma partição é criado um nó que conecta os subgrafos correspondentes. Assim, os paralelizadores trabalham de forma independente mas precisam de coordenação no momento de inserir uma requisição endereçada para mais de uma partição. Por fim, outro conjunto de *threads*, os executores, acessa o grafo de conflitos para execução das requisições cujos conflitos já foram resolvidos. Uma requisição r estará pronta para execução quando todas as requisições conflitantes entregues antes de r já tiverem sido executadas.

Seguindo esta ideia, o escalonamento híbrido adota o seguinte modelo de execução nas réplicas:

- $m + s + 1$ *threads*: um classificador, s paralelizadores e m executores.
- Cada paralelizador tem uma fila de escalonamento separada e remove requisições desta fila em ordem FIFO.
- Cada partição possui um conjunto de executores e um paralelizador associado, os quais acessam o subgrafo de dependências da partição para inserir (paralelizador) e obter/remover (executores) requisições.
- O classificador entrega as requisições na ordem total e encaminha cada requisição r para uma ou mais filas de paralelizadores:
    a. Caso r acesse apenas uma partição p, r depende das requisições anteriores endereçadas a p e é incluída na fila do paralelizador correspondente a p.
    b. Caso r acesse mais de uma partição, r depende das requisições anteriores endereçadas a estas partições e é incluída nas filas correspondentes. Neste caso, todos os paralelizadores envolvidos em r devem incluir as dependências de r em suas respectivas partições, sendo que um deles também insere a requisição em seu subgrafo. Uma requisição só é considerada inserida no grafo quando todos os paralelizadores envolvidos computarem suas dependências.
- Cada executor seleciona uma requisição que esteja pronta para execução no subgrafo correspondente, marcando-a como "em execução" e, após seu processamento, remove-a do subgrafo. Esta remoção é apenas lógica, sendo que a remoção física é realizada com a ajuda do paralelizador [Escobar et al. 2019].

2.3. Blockchains

A rápida ascensão do mercado de cripto-ativos, desde o surgimento de seu precursor e mais proeminente membro, o *Bitcoin* [Nakamoto 2008], trouxe a baila o termo *blockchain* (corrente de blocos) que vem sendo utilizado para denominar conceitos diversos como estruturas de dados e modelo de sistemas.

Estrutura de dados: uma *blockchain*, como estrutura de dados, tipicamente denomina uma lista encadeada de blocos de dados, ligados por ponteiros criptográficos (*hash pointers*). Novos dados são agrupados formando um novo bloco que é ligado através de um ponteiro de *hash* ao final da cadeia de blocos pré-existente. O bloco mais recente aponta para o bloco imediatamente anterior (Figura 1), e assim por diante, até chegar no primeiro bloco, o bloco gênesis (origem da cadeia). Os ponteiros de *hash* tanto identificam um bloco unicamente, quanto apresentam o seu *hash* criptográfico. O valor desse *hash* serve para verificar se houve ou não alterações nos dados referenciados. Como os blocos são encadeados por esses ponteiros, não é possível alterar, incluir ou excluir qualquer informação da cadeia de blocos, sem que a modificação seja facilmente detectada. Por tratar-se de uma estrutura que visa garantir a imutabilidade dos dados, essas estruturas costumam ser do tipo *append-only*, ou seja, as únicas operações permitidas numa *blockchain* são a leitura e a adição de novos blocos de dados.

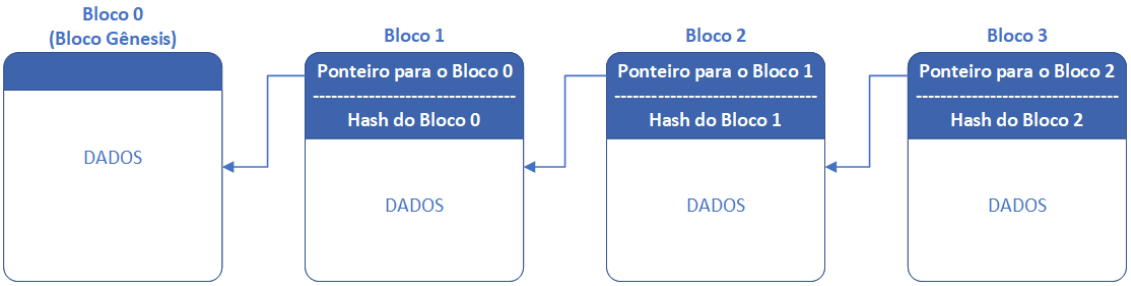

Figura 1. Encadeamento de blocos.

Modelo de sistema: como modelo de sistema, uma *blockchain* costuma ser um sistema distribuído, P2P, onde os pares colaboram entre si, seguindo um protocolo predefinido, a fim de evoluir em consenso e de forma sincronizada uma corrente de blocos. Uma característica típica desses sistemas é a independência operacional dos pares. Ou seja, mesmo cooperando com os outros nós da rede, cada nó mantém e evolui de forma autônoma, e privativa, sua própria cópia integral da cadeia de blocos, desde o bloco inicial (gênesis). O protocolo de comunicação é que garante a consistência na evolução da corrente de blocos e a correção de possíveis desvios, caso ocorram. Essa resiliência é uma das propriedades mais destacadas dos sistemas *blockchain*. O Bitcoin [Nakamoto 2008], por exemplo, possui milhares de nós ativos, porém, mesmo que 99% desses nós sofressem falha total simultaneamente, o sistema continuaria operando com os nós restantes.

2.3.1. Classificação dos sistemas blockchain

Com base na maneira como a *blockchain* evoluiu nos últimos anos, ela pode ser dividido em várias categorias com atributos distintos, embora às vezes parcialmente sobrepostos [Bashir 2017]. Aqui vamos apresentar apenas duas categorias que consideramos as principais.

Blockchains Públicas. As *blockchains* públicas, também chamadas de não permissionadas, não pertencem a ninguém. Elas são abertos ao público e qualquer pessoa pode

participar como um nó no processo de tomada de decisão, ou sair da rede a qualquer momento, sem qualquer restrição. Uma vez conectado à rede, o novo nó receberá de seus pares todos os dados que precisa para participar ativamente do protocolo em vigor. Esses sistemas em regra utilizam uma criptomoeda subjacente para criar incentivos econômicos que motivem os operadores a dedicar recursos para a sustentação da própria rede. O Bitcoin foi o primeiro e é o maior exemplo de *blockchain* pública da atualidade, com uma média de 10.000 nós conectados nos anos de 2019, 2020 e 2021 [BitNodes 2021]. A maioria das *blockchains* públicas têm por objetivo a manutenção da própria criptomoeda subjacente, como o Bitcoin [Nakamoto 2008]. Porém, outros utilizam a criptomoeda como instrumento de incentivo econômico para a manutenção da rede e regulação do uso de seus serviços, como é o caso do Ethereum [Wood et al. 2014].

Blockchains Privadas. As *blockchains* privadas estão abertas apenas para um grupo de indivíduos ou organizações que decidiram compartilhar a corrente de blocos entre si. A entrada de participantes é controlada. Novos nós precisam ser autorizados, de alguma maneira, para que possam participar da rede e ter acesso aos dados já produzidos. Cada plataforma privada controla a entrada de novos participantes à sua maneira. As implementações dessas *blockchains* são bastante heterodoxas. Existem implementações que partem de uma plataforma pública e agregam algumas funções úteis ao mundo empresarial, como é o caso do Quorum [Baliga et al. 2018] e do MultiChain [Greenspan 2015]. Outras implementações são completamente independentes como o Hyperledger Fabric [Cachin et al. 2016] e o Tendermint [Buchman 2016].

2.4. Sistemas de pagamentos

Economicamente falando, um sistema de pagamento consiste em um conjunto de instrumentos, procedimentos e, normalmente, sistemas de transferência interbancária de fundos que garantem a circulação de dinheiro [on Payment and Systems 2003]. Os sistemas de pagamento são normalmente baseados em um acordo entre os participantes e o operador do acordo, no qual todos os participantes devem confiar. Tecnologicamente falando, um sistema de pagamento é um sistema computacional que consegue gerenciar a representação digital de valores, garantindo sua propriedade exclusiva e a possibilidade de transferência dessa propriedade.

Décadas atrás, a solução encontrada pelos mercados para viabilizar a movimentação digital de valores financeiros foi a intermediação. Ou seja, as instituições financeiras ocupam o papel de intermediário (ou operador confiável) numa transação entre duas partes, garantindo para o recebedor e para o pagador que a transação seja confirmada dentro dos parâmetros acordados e que o valor transferido não possa ser reutilizado pelo mesmo pagador em outra transação (gasto duplo). Essa intermediação está baseada num sistema de confiança, onde as partes envolvidas em uma transação eletrônica devem confiar no mesmo operador financeiro, ou seus operadores financeiros devem confiar num terceiro operador financeiro comum, e assim por diante, para que a transação seja consolidada. Esse mecanismo de fato viabilizou a movimentação digital de valores, mas trouxe consigo custos e dificuldades [Burgos and Batavia 2018], por exemplo, custos operacionais, custos de reconciliação, baixa transparência para as partes, alto tempo de confirmação – principalmente quando falamos de transações internacionais ou em países

com o sistema financeiro pouco desenvolvido, – e a possibilidade do cancelamento mesmo contra a vontade das partes.

O Bitcoin foi o primeiro sistema de pagamentos[2] eletrônico desintermediado que se tem notícia [Nakamoto 2008]. Sua maior contribuição foi a forma prática com a qual resolveu o problema do gasto duplo, utilizando uma *blockchain* associada a um protocolo específico. O problema do gasto duplo diz respeito à incapacidade de se garantir a transferência e a propriedade exclusiva de um artefato digital simultaneamente. O papel-moeda é uma representação física, ou token físico, de um determinado valor em uma determinada moeda. Quando usamos uma cédula para pagar por algum produto, ou serviço, o valor que ela representa é transferido juntamente com sua posse para o vendedor. Para gastar o valor representado pela mesma cédula mais de uma vez, seria preciso copiá-la, o que é ilegal e passível de detecção devido às tecnologias utilizadas na produção do numerário físico. Já a cópia de um artefato digital é indistinguível de seu original, permitindo assim, a circulação indetectável de um número potencialmente ilimitado de cópias do mesmo artefato.

2.4.1. Limitações de desempenho

A tecnologia *blockchain* tem o potencial de melhorar a vida das pessoas de várias formas, inclusive por meio de sistemas de pagamentos mais baratos e eficientes (sua aplicação mais fundamental, vide Bitcoin [Nakamoto 2008]). Contudo, a adoção dessa tecnologia em sistemas de alto impacto social tem sido adiada devido as suas limitações atuais. Dentre essas limitações encontra-se em destaque o baixo desempenho.

Nas redes de *blockchain* com consenso baseado em prova de trabalho – *PoW* [Nakamoto 2008] – (e.g. Bitcoin e Ethereum), percebe-se um alto tempo de confirmação das transações (finalidade) e uma baixa taxa de processamento de transações por segundo, independentemente do número de participantes. Por exemplo, no Bitcoin este tempo é configurado para 10 minutos por bloco, sendo necessário 6 blocos para garantir com alta probabilidade que as transações não serão desfeitas. Já nas redes com consensos derivados do Paxos [Lamport 1998, Lamport et al. 2001], baixos tempos de finalidade e altas taxas de processamento só são encontradas quando se mantém um reduzido número de participantes no consenso como são os casos do Tendermint e do Hyperledger Fabric.

Conforme já discutido, o funcionamento de uma *blockchain* possui muitas similaridades com o de uma RME, principalmente quando consideramos as *blockchains* privadas. Um trabalho recente [Bessani et al. 2020] mostra também que o modelo de execução sequencial de uma RME quando aplicado a um sistema de pagamento (moeda virtual), causa um impacto significativo em seu desempenho, visto que a execução da maioria das requisições envolve operações custosas como a verificação de assinaturas criptográficas. Neste trabalho, analisaremos como uma RME paralela pode aumentar o desempenho de uma *blockchain* através de um estudo de caso para um protótipo de sistema de pagamento (moeda virtual).

[2]Oficialmente, no Brasil, o Bitcoin não é um sistema de pagamentos, porque o token movimentado por ele é considerado um ativo digital e não dinheiro.

3. *ParallelCoin*: Moeda Digital com Execuções Paralelas

ParallelCoin é um serviço simples de carteira digital que gerencia múltiplas moedas digitais baseadas no modelo UTXO (*Unspent Transaction Output*) introduzido no *Bitcoin* [Nakamoto 2008]. Neste modelo, cada objeto (UTXO) de uma moeda também é chamado de *token* e representa uma certa quantidade dessa moeda possuída por um usuário. Os usuários possuem um par de chaves pública-privada, os tokens estão sempre atrelados a uma chave pública e o dono da chave privada correspondente é o único que poderá movimentá-los.

Existem quatro tipos de operações suportadas pelo *ParallelCoin*:

- Emissão: as emissões criam um novo *token* para o emissor, na moeda informada e só podem ser realizadas por um grupo seleto de usuários (emissores) previamente configurados.
- Transferência: as transferências consomem um conjunto de *tokens* do mesmo usuário e criam outro conjunto de *tokens* na mesma moeda, porém de outros usuários.
- Câmbio: os câmbios consomem um conjunto de *tokens* do mesmo usuário e criam outro conjunto de *tokens* em outras moedas para outros usuários.
- Consulta: uma consulta retorna os saldo de tokens do remetente nas moedas solicitadas.

A operação de consulta é composta pela chave pública do usuário e a lista de moedas nas quais deseja consultar seu saldo. As transferências requerem a chave pública do remetente, os identificadores dos tokens que serão consumidos e um conjunto de pares de chave-valor, cada um contendo uma chave pública de um usuário e a quantidade de moedas que receberá. A operação de câmbio é uma especialização da operação de transferência, sendo que no câmbio, além do par chave-valor, o remetente deve informar a moeda de cada novo token que será criado. A taxa de câmbio é configurável para cada par de moedas.

Todas as requisições precisam ser assinadas para garantir sua autenticidade e, assim, comprovar a propriedade dos fundos afetados. Os tokens são identificados unicamente por um identificador composto pelo *hash* da operação que os criou e a sua posição na lista dos tokens criados nessa operação. O sistema só executará as operações que estejam corretamente formadas e assinadas pelo dono dos tokens consumidos ou consultados. Novos tokens surgem na saída de cada operação (exceto as consultas) e permanecem ativos até serem usados como entrada de outra operação posterior, quando são consumidos. Cada token só pode ser consumido uma única vez, para evitar o problema do gasto-duplo [Nakamoto 2008]. As réplicas devem garantir essa propriedade.

Modelo de concorrência. O estado do sistema, mantido pelas réplicas do serviço, é composto pela lista de emissores autorizados e um conjunto de tabelas de tokens atribuídos a cada usuários, sendo uma tabela para cada moeda gerenciada. Desta forma, o estado do sistema é dividido em partições, sendo que cada partição armazena as informações relacionadas a uma moeda específica. O serviço do *ParallelCoin* não precisa se preocupar com problemas de concorrência, uma vez que a camada subjacente de RME paralela só permitirá a execução paralela de operações que não possuem conflitos.

Seguindo o modelo de RME paralela adotado (Seção 2.2), os clientes indicam quais moedas (partições) serão acessadas em sua requisição e o classificador despacha as requisições para os paralelizadores correspondentes, sendo que existe um paralelizador para cada moeda. Estes paralelizadores inserem as requisições em seus respectivos subgrafos. Quando as requisições que envolvem mais de uma moeda (partição), como o câmbio, os paralelizadores das partições envolvidas inserem as dependências que ligam a nova requisição ao seu subgrafo, mas só um deles insere também a requisição. Conforme já descrito, por fim um conjunto de executores buscam requisições no grafo de dependências para execução.

As operações de emissão acessam apenas uma partição (moeda) e criam os tokens correspondentes na conta especificada. Da mesma forma, as operações de transferência também acessam apenas uma partição para consumir (remetente) e criar (destinatário) tokens. Como estas duas operações modificam o estado da partição, as mesmas conflitam com qualquer outra operação dentro da mesma partição desde que envolva algum usuário em comum. Note que transferências entre diferentes usuários não são conflitantes pois alteram diferentes entradas na tabela de tokens.

Câmbios são operações globais que acessam mais de uma partição, i.e., as partições relacionadas com as moedas usadas na operação de câmbio. Estas operações alteram o estado e, com isso, conflitam com qualquer outra operação que acessa pelo menos uma das partições e que possuem pelo menos um usuário em comum.

Por fim, as consultas funcionam tanto como operações internas em uma partição, em uma única moeda, quanto como operações globais, em mais de uma moeda/partição. Duas consultas nunca conflitam, mas conflitam com qualquer uma das outras operações que alteram o estado da(s) partição(ões) acessada(s), confirme anteriormente descrito.

Algoritmo 1 Definição de conflitos

```
1:  Transaction : {
2:      type : {Consulta, Emissao, Transferencia, Cambio},   {tipo de transação}
3:      users : list of public keys,                         {usuários afetados pela transação}
4:      partitions : list of coins,                          {partições acessadas pela transação}
5:      operation : array of bytes,                          {dados da operação a ser realizada pela transação}
6:      signature : array of bytes                           {assinatura digital do remetente}
7:  }

8:  isDependent(Transaction t1, Transaction t2) : boolean
9:      if t1.type = Consulta and t2.type = Consulta then
10:         return false
11:     else
12:         return (t1.users ∩ t2.users ≠ ∅) ∧ (t1.partitions ∩ t2.partitions ≠ ∅)
```

O Algoritmo 1 apresenta a função de definição de conflitos entre duas transações. Esta função deve ser fornecida para a camada de RME paralela, possibilitando que a mesma controle os acessos ao estado da aplicação. A estrutura $Transaction$ possui todos os dados relacionados com a transação, como o tipo de transação, os usuários afetados pela transação, as partições/moedas que a transação acessa, os dados da operação a ser

realizada e uma assinatura digital para garantir a autenticidade da transação. A função $isDependent(t_1, t_2)$ recebe duas transações como parâmetro e retornará verdadeiro caso elas conflitem, caso contrário retornará falso.

Implementação. O *ParallelCoin* foi implementado em JAVA, replicado usando o BFT-SMART [Bessani et al. 2014] e o algoritmo de escalonamento híbrido descrito na Seção 2.2. Os clientes geram operações/transações assinadas, rotulam suas operações com as partições adequadas e as enviam para o protocolo de multicast atômico do BFT-SMART. Este protocolo executa um consenso bizantino para ordenar um lote de operações por vez. Portanto, cada réplica entrega os mesmos lotes de requisições na mesma ordem total. Todas as implementações desenvolvidas estão livremente disponíveis no seguinte repositório: `https://github.com/aldenioburgos/library/tree/hibrid`.

4. Experimentos

Esta seção apresenta os experimentos realizados com o objetivo de analisar o desempenho do *ParallelCoin*. Para fins de comparação, também implementamos uma versão sequencial do sistema, que é replicada da forma tradicional sobre o BFT-SMART, i.e., todas as operações são executadas de forma sequencial.

Ambiente experimental. O ambiente experimental foi configurado com 8 máquinas conectadas a uma rede comutada de 10Gbps no Emulab [White et al. 2002]. As máquinas foram configuradas com o sistema operacional Ubuntu Linux 20.04 e uma máquina virtual Java de 64 bits versão 15.0.2. O BFT-SMART foi configurado com 4 réplicas hospedadas em máquinas separadas (Dell PowerEdge R820 equipados com quatro processadores Intel Xeon E5-4620 com 8 núcleos e 16 *threads* executando a 2,2 GHz com 128 GB de RAM) para tolerar a falha bizantina de até uma réplica. Adicionalmente, 2400 clientes foram distribuídos uniformemente em outras 4 máquinas (Dell PowerEdge R430 equipados com dois processadores Intel E5-2630v3 de 8 núcleos e 16 *threads* executando a 2,4 GHz com 64 GB de RAM).

Cargas de trabalho e configurações. Antes de cada experimento, o sistema foi pré-carregado com 2400 usuários (chaves públicas), e cada usuário com 10 tokens de valor 1 em cada uma das moedas. A taxa de câmbio entre as moedas foi configurada como 1 para todos os pares de moedas. Nos testes de desempenho, após cada transferência ou câmbio o sistema retorna para seu estado inicial, assim os tokens consumidos voltam para o estado ativo e os novos tokens gerados são descartados. Essa configuração permitiu que cada *thread* cliente executasse um número potencialmente ilimitado de requisições só com seus 10 tokens iniciais. O algoritmo de *hash* utilizado foi o *SHA256*, enquanto que o algoritmo de assinatura digital escolhido foi o algoritmo de assinatura digital de curvas elípticas (*ECDSA*) com o algoritmo de *hash SHA256*.

Dois conjuntos de experimentos foram executados, e os resultados da média do *throughput* obtido nos servidores são reportados nas figuras 2 e 3. O primeiro conjunto é composto por 90% de consultas e 10% de transferências, sendo que todas as operações

envolvem apenas uma partição (moeda) por vez. No segundo, temos 90% de consultas e 10% de movimentações (transferências ou câmbios), sendo que 5% das movimentações são câmbios envolvendo duas moedas e o mesmo percentual das consultas também envolve duas moedas. A cada requisição, a *thread* cliente sorteia (seguindo uma distribuição uniforme) o tipo da operação, o usuário receptor e as moedas envolvidas, cria uma nova operação e assina digitalmente antes de enviar para o servidor. Todas as movimentações envolveram exatamente dois usuários, o pagador e o recebedor. Conforme já comentado, as operações envolveram no máximo duas moedas.

Resultados. A Figura 2 apresenta o *throughput* obtido tanto pela execução sequencial quanto por diversas combinações de número de partições/moedas e *threads* executores na solução que permite execuções paralelas. Perceba que na configuração com apenas uma partição e um executor por partição também temos uma execução sequencial, enquanto que em todos os demais casos temos execuções paralelas. É possível identificar com clareza que o desempenho da RME paralela supera por uma larga margem o da RME sequencial. Enquanto que o *throughput* na RME sequencial manteve-se em aproximadamente $4k$ operações por segundo, a solução paralela escalou com o número de partições e conseguiu atingir um valor superior a $55k$ operações por segundo na configuração com 8 partições e 4 executores por partição (32 no total). De fato, as soluções para RME paralela conseguem aumentar o desempenho na medida que a carga de trabalho oferece mais oportunidade de paralelismo (por exemplo, aumentando o número de partições/moedas) ou mais *threads* são disponibilizadas para executar a mesma carga de trabalho.

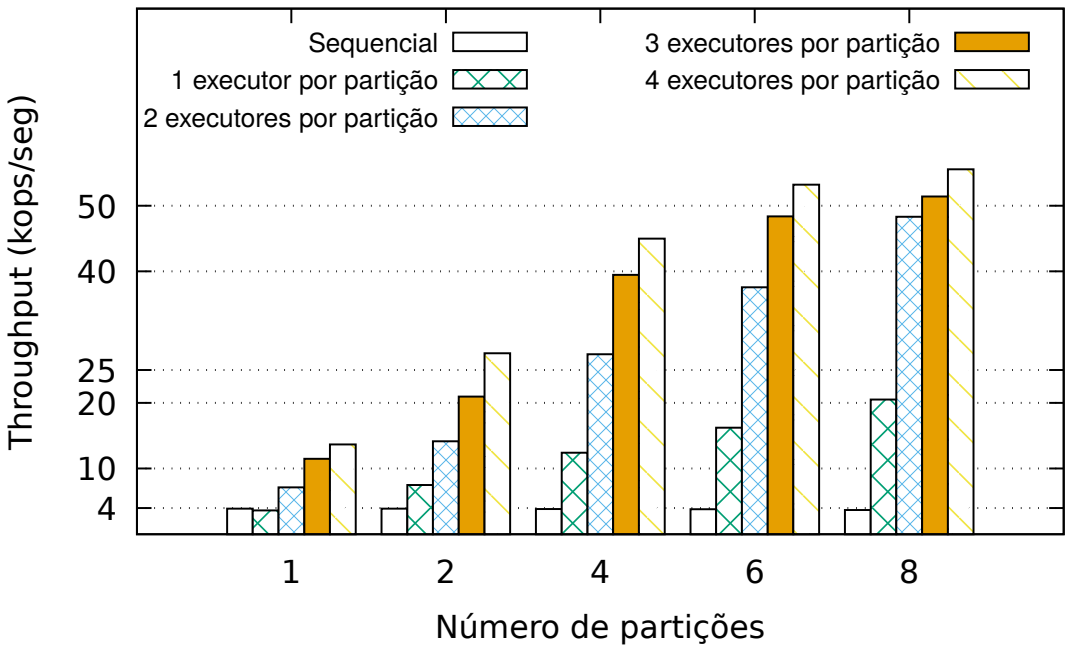

Figura 2. Throughput para diferentes números de partições e executores para uma carga de trabalho com 90% de consultas e 10% de transferências.

A Figura 3 mostra resultados similares para a segunda carga de trabalho. Como esta carga de trabalho contém operações que acessam mais de uma partição, não faz sen-

tido executar o caso com apenas uma partição, que é idêntico ao apresentado na Figura 2. O desempenho da RME sequencial continua limitado pela velocidade de processamento de uma única *thread* e novamente manteve-se próximo de $4k$ operações por segundo. Por outro lado, a implementação com RME paralela apresentou um desempenho largamente superior, aproximando-se de $30k$ operações por segundo em algumas configurações.

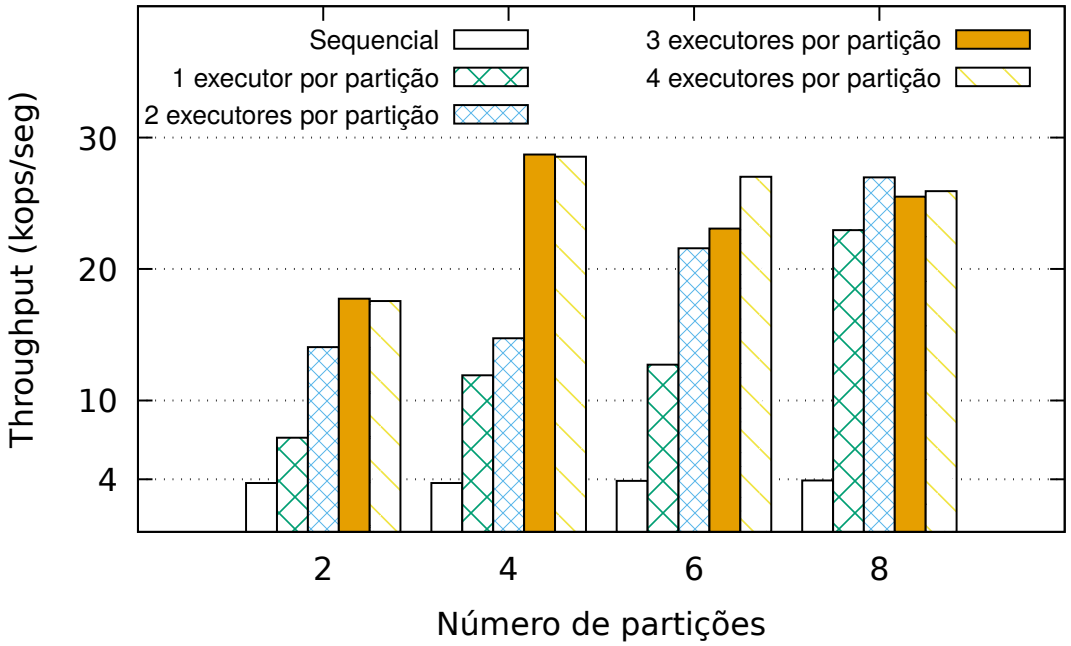

Figura 3. Throughput para diferentes números de partições e executores para uma carga de trabalho com 90% de consultas e 10% de movimentações (transferências/câmbios), sendo que 5% de todas as operações envolvem duas moedas/partições e 95% só uma moeda/partição.

5. Conclusões

Através de uma aplicação de moeda digital, um trabalho recente [Bessani et al. 2020] mostrou que o modelo de execução sequencial de RMEs tradicionais é um fator que afeta significativamente o desempenho do sistema, quando usamos este mecanismo como um bloco de construção na implementação de *blockchains*. O estudo aqui apresentado também utiliza uma aplicação de moeda digital, chamada de *ParallelCoin*, para demostrar que RMEs paralelas conseguem minimizar as limitações de desempenho anteriormente identificadas, pois permitem que uma parte das transações seja executada em paralelo nas réplicas, seguindo um modelo de concorrência elaborado para as operações definidas para a aplicação.

Experimentos realizados com as implementações desenvolvidas mostram que o *ParallelCoin* possui um desempenho de até $13.75\times$ o da solução sequencial, quando consideramos cargas de trabalho com operações que acessam apenas uma partição/moeda. Já para cargas de trabalho com operações endereçadas a mais de uma partição/moeda, o desempenho do *ParallelCoin* é até $7\times$ maior do que a solução com apenas execuções sequenciais.

Referências

Abraham, I., Malkhi, D., Nayak, K., Ren, L., and Spiegelman, A. (2017). Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus. In *Proceedings of the 21st International Conference on Principles of Distributed Systems*, Lisboa, Portugal.

Alchieri, E., Dotti, F., Mendizabal, O. M., and Pedone, F. (2017). Reconfiguring parallel state machine replication. In *SRDS*.

Alchieri, E., Dotti, F., and Pedone, F. (2018). Early scheduling in parallel state machine replica. In *ACM SoCC*.

Baliga, A., Subhod, I., Kamat, P., and Chatterjee, S. (2018). Performance evaluation of the quorum blockchain platform. *arXiv preprint arXiv:1809.03421*.

Bashir, I. (2017). *Mastering blockchain*. Packt Publishing Ltd.

Bessani, A., Alchieri, E., Sousa, J., Oliveira, A., and Pedone, F. (2020). From byzantine replication to blockchain: Consensus is only the beginning. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 424–436.

Bessani, A., Sousa, J., and Alchieri, E. E. (2014). State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362.

BitNodes (2021). Global bitcoin nodes distribution.

Buchman, E. (2016). *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis.

Burgos, A., Alchieri, E., Dotti, F., and Pedone, F. (2021). Replicação máquina de estados paralelas com escalonamento híbrido. In *Anais do 39º Simpósio Brasileiro de Redes de Computadores - SBRC 2021*. SBC.

Burgos, A. and Batavia, B. (2018). O meio circulante na era digital. *Banco Central do Brasil*.

Cachin, C. et al. (2016). Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*.

Cachin, C. and Vukolic, M. (2017). Blockchain consensus protocol in the wild (invited paper). In *Proceedings of the 31th International Symposium on Distributed Computing*, Vienna, Austria.

Castro, M. and Liskov, B. (1999). Practical Byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pages 173–186. USENIX Association.

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

Défago, X., Schiper, A., and Urbán, P. (2004). Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421.

Escobar, I. A., Alchieri, E., Dotti, F. L., and Pedone, F. (2019). Boosting concurrency in parallel state machine replication. In *Proc. 20th International Middleware Conference*.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382.

Golan-Gueta, G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M. K., Seredinschi, D., Tamir, O., and Tomescu, A. (2019). SBFT: a scalable decentralized trust infrastructure for blockchains. In *Proceedings of the 49th IEEE/IFIP International Conference on Dependable Systems and Networks*, Portland, OR, USA.

Greenspan, G. (2015). Multichain private blockchain-white paper. *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf*.

Guerraoui, R., Hamza, J., Seredinschi, D.-A., and Vukolic, M. (2019). Can 100 machines agree? *CoRR*, abs/1911.07966.

Hadzilacos, V. and Toueg, S. (1993). Fault-tolerant broadcasts and related problems. In Mullender, S., editor, *Distributed Systems*, pages 97–145. ACM Press/Addison-Wesley, New York, NY, USA.

Herlihy, M. and Wing, J. M. (1990). Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programing Languages and Systems*, 12(3):463–492.

Kotla, R. and Dahlin, M. (2004). High throughput byzantine fault tolerance. In *IEEE/IFIP Int. Conference on Dependable Systems and Networks*.

Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169.

Lamport, L. et al. (2001). Paxos made simple. *ACM Sigact News*, 32(4):18–25.

Liu, J., Li, W., Karame, G. O., and Asokan, N. (2019). Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Transactions on Computers*, 68(1):139–151.

Nakamoto, S. (2008). A peer-to-peer electronic cash system. *Bitcoin. – URL:https://bitcoin.org/bitcoin.pdf*.

on Payment, C. and Systems, S. (2003). A glossary of terms used in payments and settlement systems.

Pass, R. and Shi, E. (2017). Hybrid Consensus: Efficient Consensus in the Permissionless Model. In *Proceedings of the 31st International Symposium on Distributed Computing*, Vienna, Austria.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. (2002). An integrated experimental environment for distributed systems and networks. In *Symposium on Operating Systems Design and Implementation*.

Wood, G. et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32.