PUC

ISSN 0103-9741

Monografias em Ciência da Computação

nº 06/2021

Estudo de Algoritmos Distribuídos na Prática: Exemplos e Comparações

Lucas Borges

Vinicius Pereira

Pedro Magalhães

Fernando Lima

Markus Endler

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 06/2021 ISSN: 0103-9741
Editor: Prof. Carlos José Lucena Outubro, 2021

Estudo de Algoritmos Distribuídos na Prática: Exemplos e Comparações

Lucas Borges, Vinicius Pereira, Pedro Magalhães, Fernando Lima, Markus Endler

lucasrborges94@gmail.com, viniciusgomespe@gmail.com, pfsmagalhaes@gmail.com, folivelima@gmail.com, endler@inf.puc-rio.br

Abstract This text presents examples of the use of distributed algorithms in several application areas - to solve very specific problems of coordination and communication between distributed instances - followed by a discussion of the advantages and limitations of the solutions. We hope that this tech report will contribute to a broader understanding of interesting distributed problems and the proposed solutions, as well as to a better understanding of the possible applications. And the relevance and actuality of these algorithms in the realization of global computational systems, increasingly connected and complex.

Keywords: broadcast epidêmico; redução de dimensionalidade; jogos distribuidos; consenso em blockchain

Resumo Esse texto apresenta exemplos do uso de algoritmos distribuídos em diversas áreas de aplicação - para resolver problemas bem específicos de coordenação e comunicação - entre instâncias distribuídas, seguidos de uma discussão sobre os as vantagens e limitações das soluções. Esperamos que essa monografia contribua para um entendimento mais amplo de problemas distribuídos interessantes e as soluções propostas, bem como uma maior compreensão das possíveis aplicações. E a relevância e atualidade desses algoritmos na realização de sistemas computacionais globais, cada vez mais conectados e complexos.

Palavras-chave: Epidemic Braoadcast; distributed processing in dimensionality reduction; latency reduction in first person shooting game; consensus in blockchain

Encarregado das publicações
PUC-Rio Departamento de Informática - Publicações
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: publicar@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Sumário

1 Prefácio

Com o advento das redes de computadores nos anos 60-70, cresceu também o interesse por Algoritmos Distribuídos, área do conhecimento que desde então se tornou cada vez mais relevante, sobretudo para o projeto, a implementação, testes e validação de sistemas distribuídos e em rede. Nestes sistemas, a comunicação entre nós em rede é sujeita à falhas e imprevisível, podendo apresentar atrasos, entrega das mensagens fora de ordem, e até perda de mensagens. Além disso, existe a possibilidade de alguns dos nós da rede falharem independente- e silenciosamente. É nesse contexto de incertezas acerca da comunicação entre nós que algoritmos distribuídos precisam operar de forma coordenada, e assim garantir certas propriedades globais de funcionamento, como por exemplo, a exclusividade de um nó com no papel de coordenador, a inviolabilidade de um invariante, a alta disponibilidade do sistema como um todo, a convergência coletiva para um certo estado dos nós, escalabilidade, etc. Atualmente, com o crescente uso da computação em nuvem para processamento e análise de dados, da mobilidade de dispositivos portáteis de usuários, da computação pervasiva/ubíqua e da Internet das Coisas, notou-se a necessidade de conhecer melhor, utilizar e adaptar algoritmos distribuídos tradicionais para tais sistemas mais dinâmicos e móveis, tanto no nível de middleware como no nível das aplicações distribuídas.

Além disso, as características inéditas e dos requisitos desafiadores desses novos sistemas, e sobretudo dos novos tipos de rede utilizados (i.e. maior volatilidade e dinamismo das conexões par-a-par), levaram a necessidade de criar novos algoritmos distribuídos que garantissem a consistência e as propriedades da aplicação também nesses novos contextos de execução e de redes móveis.

Porém, a despeito das diferenças entre os diferentes sistemas, redes e dos novos requisitos sobre aplicações distribuídos, alguns princípios fundamentais permanecem e podem ser encontrados em vários algoritmos, como por exemplo: a opção pela garantia da consistência fraca de estados distribuídos em prol da escalabilidade, a convergência eventual das propriedades requeridas pela aplicação, a definição de *detectores de falha imprecisos* com diferentes garantias, temporizadores que se adaptam às condições momentâneas da rede, processamento coletivo em rodadas/etapas assíncronas, etc.

Essa monografia apresenta algumas aplicações - naturalmente distribuídas - com seus problemas específicos de manutenção de um estado global consistente e satisfazendo as propriedades globais requeridas pela aplicação em uma topologia de rede dinâmica e variável. E como o emprego de alguns algoritmos distribuídos específicos pode resolver tais problemas, seguido de uma análise sobre as vantagens e limitações das soluções encontradas na literatura.

Esperamos que esse documento contribua para um entendimento mais amplo de problemas distribuídos interessantes e as soluções propostas, bem como uma maior compreensão das possíveis aplicações. E a relevância e atualidade desses algoritmos na realização de sistemas computacionais globais, cada vez mais conectados e complexos.

Boa leitura!

Markus

2 Algoritmos para Broadcast Epidêmico

por Lucas Borges

2.1 Contexto

Broadcast é um componente básico da implementação de aplicações e sistemas distribuídos. Essa primitiva tem o objetivo de garantir que todos os participantes do sistema funcionando corretamente recebam todas as mensagens de *broadcast*, mesmo na presença de falhas de rede ou de nós.

Devido à ubiquidade dessa primitiva, algoritmos escaláveis e confiáveis são necessários. Esses algoritmos podem ser divididos em duas categorias: probabilísticos e de consenso. Esse capítulo irá focar na primeira, também conhecida como algoritmos epidêmicos ou de rumores.

2.2 Área de aplicação

Algoritmos de *broadcast* são fundamentais em inúmeras aplicações distribuídas, por exemplo: notificação, distribuição de conteúdo, replicação e comunicação em grupo. Para melhor ilustrar sua relevância e abrangência, consideremos replicação em mais detalhe.

Replicação consiste em gerar redundância no sistema para aumentar sua disponibilidade, capacidade de transferência e reduzir latência. Disponibilidade refere-se ao sistema ser capaz de continuar operando mesmo com a ocorrência de falhas em certa quantidade de nós. Já a maior capacidade de transferência e redução de latência decorrem da distribuição de acesso e geográfica dos dados. Porém esses dados guardados em mais de um local precisam ser mantidos consistentes entre si, e uma maneira disso ser feito são algoritmos de *broadcast*.

O uso de replicação em serviços e aplicações distribuídos faz-se portanto bem evidente. Sejam eles pequenos, com alguns poucos servidores, ou serviços globais com milhões de servidores como o Google.

2.3 Principais Desafios sendo tratado

O objetivo de algoritmos de *broadcast* é garantir que todos o nós corretos, i.e. não falhos, recebam todas as mensagens de *broadcast*, mesmo diante de falhas em enlaces de rede ou de nós intermediários. Frente a esse objetivo, há três desafios principais a serem considerados: confiabilidade, escalabilidade e tempo de propagação.

2.3.1 Confiabilidade

Confiabilidade refere-se à garantia de que todos os nós corretos irão receber o *broadcast* e como obstáculo a esse objetivo apresentam-se as falhas. Falhas de rede podem se apresentar como perda de mensagens ou partições de rede. Já falhas de nós envolvem estes pararem de operar, temporária- ou definitivamente. Falhas bizantinas, i.e. erros de funcionamento com comportamento arbitrário, não serão consideradas.

Além das falhas, algoritmos probabilísticos por sua natureza não garantem automaticamente atomicidade. Dependendo dos parâmetros do algoritmo, uma porcentagem dos

nós ativos recebem o *broadcast*. Uma confiabilidade de 100% significa que todos os nós receberam a mensagem, ou seja, o *broadcast* foi atômico.

2.3.2 Escalabilidade

Escalabilidade refere-se ao número de mensagens gerada para propagar o *broadcast*. Idealmente o tráfego de rede gerado é proporcional ao tamanho da mensagem multiplicado pelo número de servidores, porém alguns algoritmos geram muito mais tráfego do que esse mínimo [1].

2.3.3 Tempo de propagação

Tempo de propagação refere-se ao tempo que o *broadcast* leva para entregar a mensagem a todos os nós. Pode-se definir dois valores de tempos relevantes: t_{ave} e t_{last}. O primeiro é a média do tempo que a mensagem levou para chegar em cada um dos nós. O segundo é o tempo que a mensagem levou para chegar ao último nó que não havia recebido a mensagem.

2.4 Abordagem de Solução

Algoritmos epidêmicos, como o nome sugere, têm base em teorias epidemiológicas[1]. Eles são uma abordagem interessante para implementar primitivas de *broadcast* de alta confiabilidade e escalabilidade pois possuem uma resiliência natural à perda de mensagens e falhas de nós[2].

A ideia básica desses algoritmos, inspirados na disseminação de rumores (gossip), consiste em distribuir a tarefa de espalhar a mensagem igualmente entre os nós da rede, e sendo portanto algoritmos de natureza *peer-to-peer*. Dessa distribuição de responsabilidade vem a alta escalabilidade e tolerância a falha intrínsecos à eles.

Nós são inicialmente ignorantes e quando um recebe dados a serem divulgados torna-se "infeccioso". Periodicamente esse nó escolhe outro aleatoriamente e confere se ele possui o dado novo. Se não o possuía, esse novo nó também torna-se infeccioso e repete o mesmo procedimento. Se o nó recebeu o dado previamente, simplesmente descarta a mensagem. Quando um nó tentou transmitir o dado a um certo número de nós que já o possui este para de tentar transmitir e torna-se "removido". Um nó que ainda não recebeu a mensagem é chamado de "suscetível".

Para operar como descrito acima, um nó precisaria manter informação sobre todos os outros nós que pertencem ao grupo. Além de claramente não ser algo escalável pelo alto número de nós que podem fazer parte do grupo, a escolha uniforme de outro nó pode gerar alto tráfego ao escolher um nó distante. Assim, depende-se de uma visão parcial dos nós que pertencem ao grupo, geralmente baseada na distribuição espacial dos nós. Essa preferência por nós próximos forma uma vizinhança que pode ser descrita como uma rede lógica sobreposta à física.

Devido à sua natureza randômica, algoritmos de rumores possuem uma probabilidade de a mensagem não chegar a todos os nós da rede. Essa probabilidade de falha pode ser feita arbitrariamente pequena[1], mas é importante estudá-la cuidadosamente com análise e simulações.

Infelizmente, em estado estável, algoritmos epidêmicos demonstram um *overhead* de mensagens excessivo para garantir confiabilidade com alta probabilidade. Por outro lado,

algoritmos baseados em árvores possuem uma complexidade de mensagens baixa em estado estável, mas são frágeis na presença de falhas [2]. Multicast bimodal [3] foi um dos primeiros a combinar ambos numa abordagem híbrida mas possui certas desvantagens ao requerer dois protocolos diferentes[2]. Plumtree é outro algoritmo que combina ambos para atingir uma abordagem com baixa complexidade de mensagens e alta confiabilidade[2].

2.5 Algoritmos Epidêmicos

2.5.1 Algoritmo epidêmico de Demers

O artigo de Demers et al.[1] explora diferentes variações no algoritmo de rumores básico no contexto de um banco de dados altamente replicado e seus impactos na velocidade de convergência da rede. Os algoritmos explorados dependem somente da eventual entrega de mensagens repetidas. Eles afirmam que o uso de protocolos randomizados permitem que os algoritmos tenham uma implementação direta usando estruturas de dados simples e que a probabilidade da informação não ter convergido é exponencialmente decrescente no tempo.

Segundo a literatura epidemiológica, o espalhamento de rumores pode ser modelado com um par de equações diferenciais. Seja s, i e r as frações de indivíduos suscetíveis, infecciosos e removidos respectivamente. Seja $1/k$ a probabilidade de um individuo infeccioso se tornar removido quando entra em contato com um nó já infectado. Temos:

$$\frac{ds}{dt} = -si$$

$$\frac{di}{dt} = +si - \frac{1}{k}(1-s)i$$

Resolvendo para i em função de s:

$$i(s) = \frac{k+1}{k}(1-s) + \frac{1}{k}\log s$$

Essa função é zero quando:

$$s = e^{-(k+1)(1-s)}$$

Que mostra que s diminui exponencialmente com k. Portanto, aumentando k garantimos que quase todos os nós recebam o rumor.

A partir disso, exploram diferentes variações do algoritmo de rumores. São comparados algoritmos cegos e com *feedback*; com contadores e probabilísticos; e algoritmos *push* com *pull*. Para efetuar essas comparações, três métricas são definidas: resíduo, tráfego e tempo de propagação.

Resíduo é qual fração de nós ainda permanecem suscetíveis quando a epidemia termina, ou seja, não há mais nós infecciosos. Tráfego é a média do número de mensagens enviadas por todos os nós. Tempo de propagação se refere a t_{ave} e t_{last}.

O algoritmo descrito previamente utilizava o *feedback* do recipiente. Um nó infeccioso só possuía chance de se tornar removido se quem ele entrasse em contato já havia recebido a mensagem. A variação cega torna-se removido com probabilidade $1/k$ independentemente do destinatário, ou seja, não precisa de uma resposta.

Tabela 1: Performance de uma epidemia em 1000 nós usando *feedback* e contadores

Contador k	Resíduo s	Tráfego m	Tempo de propagação t_{ave}	t_{last}
1	0.176	1.74	11.0	16.8
2	0.037	3.30	12.1	16.9
3	0.011	4.53	12.5	17.4
4	0.0036	5.64	12.7	17.5
5	0.0012	6.68	12.8	17.7

Tabela 2: Performance de uma epidemia em 1000 nós usando contato cego e probabilidade

Probabilidade k	Resíduo s	Tráfego m	Tempo de propagação t_{ave}	t_{last}
1	0.960	0.04	19	38
2	0.205	1.59	17	33
3	0.060	2.82	15	32
4	0.021	3.91	14.1	32
5	0.008	4.95	13.8	32

Uma alternativa a tornar-se removido com probabilidade $1/k$ é utilizar um contador. Um nó deixaria de ser infeccioso depois de k contatos. Isso pode ser combinado tanto com a variação cega quanto com a que utiliza *feedback*.

De acordo com simulações executadas por seu time, a variação de contadores com *feedback* possuem o melhor tempo de propagação, com contadores tendo mais impacto sobre o resultado do que *feedback*.

Até agora vimos somente epidemias *push*: cada nó infeccioso escolhe aleatoriamente um outro nó para tentar infectar. Na variação *pull* cada nó, independentemente de seu estado, entra em contado com outro aleatório e confere se ele possui alguma mensagem que não conhece ainda. A principal diferença no comportamento das duas variações depende da frequência de mensagens que são introduzidas na rede. Se existem vários *broadcasts* independentes sendo introduzidos na rede, *pull* tem alta chance de encontrar um nó com uma mensagem nova. Porém, se a rede está quieta, *push* não gera continuamente requisições desnecessárias

Algoritmos epidêmicos conseguem realizar o *broadcast* de uma mensagem rapidamente e com baixo tráfego. Porém, mesmo que conseguimos fazer a chance dele falhar ser extremamente baixa, a probabilidade não é zero. Para eliminar essa possibilidade os autores propuseram uma solução que era conveniente para o contexto que estavam investigando:

Tabela 3: Performance de uma epidemia *pull* em 1000 nós usando *feedback* e contadores.

Contador k	Resíduo s	Tráfego m	Tempo de propagação t_{ave}	t_{last}
1	3.1 x 10-2	2.70	9.97	17.63
2	5.8 x 10-6	4.49	10.07	15.39
3	4.0 x 10-6	6.09	10.08	14.00

utilizaram um algoritmo anti-entropia como um mecanismo *backup*. Esse algoritmo mais caro seria rodado com menor frequência na rede para assegurar com probabilidade 1 que toda mensagem eventualmente chega a todos os nós.

2.5.2 Algoritmo de Plumtree

Algoritmos de rumores demonstram um alto *overhead* de mensagens mesmo quando a rede está bem comportada para garantir confiabilidade com alta probabilidade. Do outro lado, algoritmos de *broadcast* baseados em árvores tem baixo volume de mensagens em uma rede bem comportada mas são frágeis na presença de falhas. O algoritmo de *plumtree*[2] combina protocolos de rumores com protocolos baseados em árvores para conseguir tanto uma baixa complexidade de mensagens quanto alta confiabilidade.

Esse algoritmo cria uma árvore de *broadcast* embutida na rede lógica criada pelo algoritmo de rumores. Essa árvore é criada e mantida usando um algoritmo de baixo custo, reagindo a falhas na rede e se regenerando quando necessário. Para realizar o *broadcast*, os autores utilizam a estratégia de *push* e a subdividem em dois tipos: *eager push* e *lazy push*.

Em *eager push* nós enviam a mensagem completa para nós selecionados aleatoriamente assim que a recebe. Já em *lazy push* o nó envia somente o identificador da mensagem a princípio, se o destinatário ainda não recebeu a mensagem correspondente ele faz uma requisição e o nó envia o conteúdo da mensagem.

Plumtree realiza o *broadcast* primariamente utilizando *eager push* pelas arestas da árvore gerada. Contudo, o resto da rede lógica de rumor também é utilizada para propagar a mensagem utilizando *lazy push*. Dessa maneira, em estado estável da rede o volume de mensagens é reduzido, já que grande parte delas são apenas identificadores de tamanho pequeno ou agrupados. Enquanto que na presença de falhas o protocolo utiliza a resiliência intrínseca da abordagem por rumores, com a diferença que os nós contatados não mudam a cada rodada.

O algoritmo depende de um serviço que gere a rede lógica baseada em rumores e notifique de alterações de membros da visão parcial de cada nó para manter sua árvore. O *Plumtree Protocol* é o componente que tem como responsabilidade as operações de construir e consertar a árvore utilizando esse serviço. O *overhead* dessas operações deve ser o menor possível em termos de mensagens de controle. O processo de conserto deve garantir que mesmo na presença de falhas todos os nós sejam cobertos pela árvore, ou seja, precisa detectar e reparar partições.

Esse serviço deve garantir algumas propriedades sobre a rede lógica gerada. A rede lógica precisa ser conexa independente de falhas: todo nó precisa ter no mínimo um outro nó correto em sua visão e todo nó deve estar na visão de pelo menos um nó correto. O serviço deve ser escalável e operar corretamente em grandes redes. O serviço deve minimizar alterações na visão dos nós quando operando em um estado estável.

O conjunto *eagerPushPeers* é inicializado com todos os nós que pertencem à vizinhança segundo o serviço da rede lógica. A árvore é construída movendo nós de *eagerPushPeers* para *lazyPushPeers* até que o primeiro seja uma árvore. Quando um nó recebe uma mensagem pela primeira vez, ele inclui o remetente em *eagerPushPeers* e garante que a conexão é bidirecional. Quando uma mensagem duplicada é recebida, o remetente é movido para *lazyPushPeers* e uma mensagem PRUNE é enviada para que ele também faça a mesma operação. Essas operações garantem que quando o primeiro *broadcast* seja feito,

```
1  procedure dispatch do
2      announcements ⟵ policy (lazyQueue) //set of IHAVE messages
3      trigger Send(announcements)
4      lazyQueue ⟵ lazyQueue \announcements

5  procedure EagerPush (m, mID, round, sender) do
6      foreach p ∈ eagerPushPeers: p ≠sender do
7          trigger Send(GOSSIP, p, m, mID, round, myself)

8  procedure LazyPush (m, mID, round, sender) do
9      foreach p ∈ lazyPushPeers: p ≠sender do
10         lazyQueue ⟵ (textscIhave(p, m, mID, round, myself)
11     call dispatch()

12 upon event Init do
13     eagerPushPeers ⟵ getPeers(f)
14     lazyPushPeerss ⟵ ∅
15     lazyQueues ⟵ ∅
16     missing ⟵ ∅
17     receivedMsgs ⟵ ∅

18 upon event Broadcast(m) do
19     mID ⟵ hash(m+myself)
20     call EagerPush (m, mID, 0, myself)
21     call lazyPush (m, mID, 0, myself)
22     trigger Deliver(m)
23     receivedMsgs ⟵ receivedMsgs ∪ {mID}

24 upon event Receive(GOSSIP, m, mID, round, sender) do
25     if mID ∉ receivedMsgs then
26         trigger Deliver(m)
27         receivedMsgs ⟵ receivedMsgs ∪ {mID}
28         if ∃ (id,node,r) ∈ missing :id=mID then
29             cancel Timer(mID)
30         call EagerPush (m, mID, round+1, myself)
31         call lazyPush (m, mID, round+1, myself)
32         eagerPushPeers ⟵ eagerPushPeers ∪ {sender}
33         lazyPushPeers ⟵ lazyPushPeers \ {sender}
34         call Optimize (m, mID, round, sender) // optional
35     else
36         eagerPushPeers ⟵ eagerPushPeers \ {sender}
37         lazyPushPeers ⟵ lazyPushPeers ∪ {sender}
38         trigger Send(PRUNE, sender, myself)

39 upon event Receive(PRUNE, sender) do
40         eagerPushPeers ⟵ eagerPushPeers \ {sender}
41         lazyPushPeers ⟵ lazyPushPeers ∪ {sender}
```

Figura 1: Algoritmo de construção da árvore.

uma árvore que minimiza latência é criada.

Para minimizar o *overhead* de mensagens, o *lazy push* utiliza mensagens IHAVE que são agrupadas antes de serem enviadas. Ao receber uma mensagem IHAVE, se o nó não recebeu a mensagem correspondente ainda, um temporizador é iniciado. Se não receber a mensagem por *eager push* antes do temporizador estourar, uma mensagem GRAFT é enviada para o remetente da IHAVE. Essa mensagem inicia a transmissão do conteúdo da mensagem e adiciona essa conexão à árvore. É importante notar que ciclos podem ser formado se vários nós se desconectarem com uma única falha e tentarem consertar a árvore simultaneamente, porém o processo de construção da árvore irá eventualmente removê-los.

```
1  upon event Receive(IHAVE, mID, round, sender) do
2      if mID ∉ receivedMsgs do
3          if ∄ Timer(id): id=mID do
4              setup Timer(mID, timeout_1)
5          missing ⟵ missing ∪ {(mID,sender,round)}

6  upon event Timer(mID) do
7      setup Timer(mID, timeout_2)
8      (mID,node,round) ⟵ removeFirstAnnouncement(missing, mID)
9      eagerPushPeers ⟵ eagerPushPeers ∪ {node}
10     lazyPushPeers ⟵ lazyPushPeers \ {node}
11     trigger Send(GRAFT,node,mID,round,myself)

12 upon event Receive(GRAFT, mID, round, sender) do
13     eagerPushPeers ⟵ eagerPushPeers ∪ {sender}
14     lazyPushPeers ⟵ lazyPushPeers \ {sender}
15     if mID ∈ receivedMsgs do
16         trigger Send(GOSSIP, sender, m, mID, round, myself)
```

Figura 2: Algoritmo de conserto da árvore.

Os autores realizaram simulações com o algoritmo *Plumtree* e comprovaram a redução de mensagens redundantes durante estado estável da rede e falhas leves, mesmo com as mensagens de controle introduzidas. Na presença de falhas graves, onde uma grande porcentagem de nós falha simultaneamente, há um aumento no número de mensagens redundantes devido ao processo de reparo da árvore. Porém o tráfego se normaliza depois de poucos ciclos.

2.6 Análise Comparativa

A pesquisa realizada por Demers Et al. tinha como objetivo a aplicação em um contexto e sistema específico da Xerox, onde partições em sua rede global eram possíveis. Para garantir a replicação completa optaram por introduzir um algoritmo mais caro como *backup* para executar com pouca frequência. Esse *backup* garantiria o *broadcast* completo eventualmente. Já a proposta do algoritmo *Plumtrees* tem como premissa que o serviço que ele utiliza provê uma rede lógica que permanece conexa meso diante de falhas graves, portanto consegue garantir a confiabilidade de 100%.

O algoritmo de *Plumtrees* faz uma abordagem híbrida do problema. Busca combinar as vantagens enquanto ameniza as desvantagens de duas abordagens diferentes. Assim, reduz o *overhead* de mensagens durante estados estáveis mas é capaz de se recuperar de um alto número de falhas enquanto mantém alta confiança[2].

2.7 Conclusão

Algoritmos de *broadcast* é uma área de grande atividade devido à sua grande abrangência de aplicação. Protocolos baseados em rumores como discutidos nesse capítulo são especialmente relevantes para aplicações decentralizadas, conhecidas também como *peer-to-peer*.

Um foco da pesquisa nessa área atualmente são métodos híbridos e estratégias de formação de vizinhança [2]. Outro foco atual trata-se de redes móveis e redes de sensores, o que traz outro desafio a ser considerado: o consumo de energia gerado.

Referências

[1] DEMERS, A.; GREENE, D.; HAUSER, C.; IRISH, W.; LARSON, J.; SHENKER, S.; STURGIS, H.; SWINEHART, D.; TERRY, D. Epidemic algorithms for replicated database maintenance. In: . PODC '87. New York, NY, USA: Association for Computing Machinery, c1987. p. 1–12.

[2] LEITAO, J.; PEREIRA, J.; RODRIGUES, L. Epidemic broadcast trees. In: . c2007. p. 301–310.

[3] BIRMAN, K. P.; HAYDEN, M.; OZKASAP, O.; XIAO, Z.; BUDIU, M.; MINSKY, Y. Bimodal multicast. *ACM Trans. Comput. Syst.*, New York, NY, USA, v. 17, n. 2, p. 41–88, May 1999.

3 Algoritmos para Processamento distribuído para Redução de Dimensionalidade

por Vinicius Pereira

3.1 Contexto

Os *tablets*, telefones celulares e diversos dispositivos portáteis são tecnologias indispensáveis na vida de um cidadão comum, e guardam uma combinação de ricas interações do usuário, possuindo uma quantidade sem precedentes de dados, muitos deles de natureza privada. Essa natureza sensível dos dados significa que há riscos e responsabilidades em armazená-los em um local centralizado [1]. Em 2020, a lei geral de proteção dos dados entrou em vigor no Brasil visando assegurar o direito à privacidade e à proteção de dados pessoais dos usuários, por meio de práticas transparentes e seguras, garantindo direitos fundamentais [2]. Dessa forma, a transparência e privacidade dos usuários é garantida por lei, impactando, por exemplos, empresas que mantinham dados dos usuários centralizados ou de forma não transparente. Onde o dado é guardado, qual informação e como é armazenado são de responsabilidade das empresas, e elas devem comunicar quando requisitado e informar o uso aos seus clientes.

A quantidade de dados gerados, consumidos, trocados e enviados na internet por usuários tem crescido anualmente. Com isso, demandas de armazenamento, de processamento e de rede têm atingido novos patamares cada ano. Em aplicações com múltiplos servidores de armazenamento de dados, muitas vezes, a troca e centralização de dados têm um custo alto, e por isso, o processamento de dados de forma distribuída tem ganhado espaço nos últimos anos.

Sendo assim, soluções para o processamento distribuído de dados se tornaram fundamentais, seja pela questão de privacidade, ou uma questão de custo. Decidimos estudar a redução de dimensionalidade de dados de modo a reduzir custos computacionais ao manipular informações com menos dimensões. Há vários algoritmos de redução de dimensionalidade e escolhemos a análise de componentes principais como objetivo de estudo, cuja implementação usual considera dados centralizados. Este trabalho tem como objetivo expor como análise de componentes principais pode ser implementada de forma descentralizada, considerando dados distribuídos que por questões de privacidade não podem ser trocados entre servidores.

3.2 Introdução

Redução de dimensionalidade é um processo de transformar dados de uma dimensão para outra de dimensionalidade menor, com o objetivo de ter uma maior interpretabilidade dos dados e ganhos computacionais, por exemplo. Há um *trade-off* de perda de informação, com ganho computacional. Segundo [3], este processo está intimamente relacionado ao conceito de compressão (com perdas) na teoria da informação. Há diversas vantagens em reduzir a dimensionalidade dos dados:

- A alta dimensionalidade requer muitas vezes alto poder computacional;

- Muitas dimensões aumentam a complexidade dos modelos. O aumento do número de *features*, aumenta as chances de *overfitting* - quando um modelo se ajusta aos seus dados de treino muito bem, porém não tem grande desempenho nos dados de teste, basicamente;

- Reduzir o número de *features* com pequenas perdas nos dados pode melhorar a performance (tanto da acuracidade do modelo, quanto no tempo e nos recursos de memória), nos modelos de predição;

- Redução de dimensionalidade pode ser usada para melhorar a interpretação dos dados, sua visualização, e assim, conseguir extrair informações.

Utilizaremos a *Principal Components Analysis* (PCA) para essa tarefa de redução de dimensionalidade. Como a maioria dos algoritmos estatísticos de otimização, PCA foi formulado com dados centralizados. Em alguns casos, os dados não estão disponíveis em um único repositório, o que torna esse abordagem não útil [4].

Iremos expor a análise de componentes principais de forma centralizada para em seguida expandir o algoritmo para uma abordagem distribuída dos dados.

Iremos propor um algoritmo PCA distribuído. Esta proposta tem como base [4], e formula que esse problema se resume a um cálculo de média distribuída das matrizes de covariância e de decomposições em valores singulares. Esse tipo de cálculo distribuído é possível por meio do protocolo de consenso chamado *Sum–Weight Gossip*.

3.3 Abordagem Centralizada para a análise de componentes principais

Principal Components Analysis (PCA) é uma transformação linear ortogonal que transforma o dado para um novo sistema de coordenadas de modo que a menor variância de alguma projeção escalar dos dados esteja na primeira coordenada (chamada de primeiro componente principal), a segunda maior variância na segunda coordenada e assim por diante [5].

Os primeiros componentes principais descrevem melhor os dados. E a utilização de PCA na redução de dimensionalidade recai em escolher um número de componentes menor que o número de *features* dos dados originais, retendo o máximo possível da informação.

As m observações de nossos dados podem ser representadas por $X \in \mathbb{R}^{d,m}$. Dessa forma cada linha de X é uma observação de dimensão d. Seja a matriz de projeção na nova base , tal que seja mapeado em $W \in \mathbb{R}^{n,d}$. Cada linha de WX tem dimensão n, em que $d > n$. Portanto, cada observação X_i , em que antes representamos por d *features*, após a transformação será representada por n *features*, havendo uma redução de dimensionalidade.

Iremos descrever também o problema de recuperação da informação que foi compactada por transformação linear. Ou seja, dado a nova representação de x na nova base, qual seria a transformação linear que transformaria essa nova representação na representação original?

Considere uma matriz secundária $U \in \mathbb{R}^{d,n}$ de modo que UWX se aproxime de X, fazendo o mapeamento inverso. Essa transformação inversa tem que se aproximar o máximo possível de X. Dessa forma, avaliaremos essa recuperação por uma determinada função de erro.

PCA é uma técnica em que tanto a compactação da informação quanto a recuperação é feita por transformações lineares, como já descrevemos, e o erro desse mapeamento inverso

é o erro quadrático.

Assim, gostaríamos de encontrar W e U de modo que o erro quadrático desse mapeamento inverso seja o menor possível:

$$\underset{W,U}{\operatorname{argmin}} \sum_{i=1}^{m} \|X_i - UWX_i\|_2^2$$

A solução do problema de minimização descrito acima tem como solução uma matriz ortogonal tal que $W = U^T$.

Como solução, W é constituído pelos autovetores da matriz de covariância da amostra:

$$C = \frac{1}{n} X X^T - \mu\mu^T$$

onde μ é a média da amostra.

Assim, PCA é uma técnica que transforma os dados originais em uma nova base ortogonal, como havíamos exposto. Há diversas outras técnicas de redução de dimensionalidade, como *Non-negative matrix factorization* (NMF), *Linear discriminant analysis* (LDA), *Autoencoders*, entre outras.

3.4 Abordagem Descentralizada para a análise de componentes principais

Nesta sessão faremos uma abordagem descentralizada para a análise de componentes principais. Primeiro, será dada uma visão geral a respeito da descentralização, para em seguida expormos o protocolo de propagação de boatos, que servirá como base para a comunicação em uma rede. Por fim, iremos expor um algoritmo de análise de componentes principais descentralizado, utilizando o protocolo de propagação por boatos.

3.4.1 Visão geral

Na abordagem descentralizada, há dois pontos importantes notórios a se definir, segundo [6]:

- O modelo de distribuição de dados, ou seja, como os dados estão distribuídos pela rede;
- Como a rede está estruturada e como o protocolo de comunicação é feito;

O modelo de distribuição de dados impacta diretamente no algoritmo de otimização. Segundo [4], há dois cenários a se considerar na distribuição dos dados. No modelo de amostras distribuídas, cada servidor possui registros distintos. Isto é, cada nó guarda uma amostra distinta $X_i \in \mathbb{R}^{D,n_i}$, de modo que as colunas de X que estão distribuídas. No modelo de coordenadas distribuídas, cada nó registra uma informação distinta sobre o mesmo registro. Cada nó só registra um subconjunto $X_i \in \mathbb{R}^{D_i,n}$, de modo que as linhas de X estão distribuídas. Em ambos os casos visamos a redução de dimensionalidade, porém como os dados estão distribuídos impactam em como projetar um algoritmo distribuído para esse caso.

Há algumas outras questões importantes sobre os dados. Segundo [1], os dados podem ser:

- Massivamente distribuídos: os dados são armazenados em um grande número de nós K. Em particular, o número de nós pode ser muito maior do que o número médio de exemplos de treinamento armazenados em um determinado nó;
- Não independentes e randômicos: os dados em cada nó podem ser extraídos de uma distribuição diferente. Os dados disponíveis localmente estão longe de ser uma amostra representativa da distribuição geral;
- Desbalanceados: nós diferentes podem variar em ordens de magnitude no número de exemplos de treinamento que possuem.

O modelo de comunicação entre os servidores podem diferir e interfere como o algoritmo distribuído pode ser implementado.

Na arquitetura em estrela, como mostrado na figura 1, há uma divisão hierárquica das tarefas, e emprega um nó central coordenador que também age como um nó de fusão [6]. Nesse tipo de arquitetura, pode-se também se beneficiar de uma computação paralela para acelerar a computação do algoritmo de PCA distribuído, utilizando processadores e memórias locais [6]. Na arquitetura em redes *Mesh*, os nós e links devem usar as mesmas funções e procedimentos, de forma que os nós compartilham computações parciais, mas não os dados, respeitando a privacidade [6].

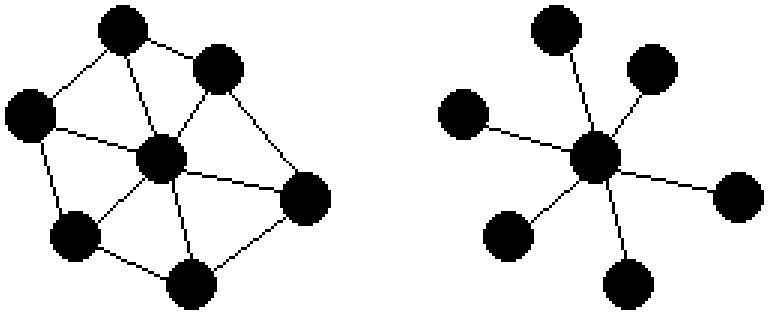


Figura 3: Arquiteturas de rede *Mesh* e Estrela, respectivamente. Em redes em estrela, os nós são divididos em um nó principal (*master*) e nós secundários (*slaves*), e a comunicação reside por meio do nó principal, que deve sempre estar disponível. Em redes *Mesh*, os nós conseguem se comunicar por mais de um caminho possível.

No artigo [4], são levantadas outras considerações sobre os algoritmos de PCA descentralizados:

- Modelo de Dados: dados de amostras não podem ser trocados de maneira alguma, por questões de privacidade ou legalidade.
- Assincronismo: os nós são assíncronos e não dependem e esperam uns aos outros;
- Descentralização: todos os nós e *links* devem atuar da mesma maneira, e os nós devem executar os mesmos procedimentos;
- Consenso: Todos os nós devem obter a mesma base ortogonal no fim do processo;

Usualmente, o número de mensagens em cada nó permitido em uma dada janela de tempo é limitada, e o custo de computação é geralmente negligenciado em comparação aos de atraso de comunicação [7]. O desempenho geralmente é investigado pelo número de mensagens enviadas. Dessa forma, a comunicação é crucialmente importante, e por isso, atraso e perda de mensagens são métricas importantes [7].

O grande atrativo de tecnologia *peer-to-peer* (P2P) para aplicações e sistemas distribuídos, em virtude da sua alta escalabilidade e seu custo baixo, e ausência de nós centrais. Por causa disso, esse tipo de tecnologia tem potencial em soluções que preservam a identidade dos dados [7]. De maneira geral, um sistema P2P consiste em um número alto de nós que se comunicam via mensagens, que podem sofrer atrasos ou serem perdidas, e os nós (*peers*) podem deixar ou sair da rede em qualquer momento. Sendo assim, esse tipo de arquitetura faz sentido no caso de uso desta monografia. Sendo assim, na próxima sessão, introduziremos o protocolo de de propagação por boatos, que será utilizado posteriormente.

3.4.2 Aprendizado por Boatos

Gossiping, ou propagação por boatos, é um protocolo de comunicação *peer-to-peer* em uma rede baseado em como uma epidemia se espalha[8]. Neste protocolo, cada nó contacta um ou poucos nós em cada etapa, geralmente escolhidos randomicamente. A dinâmica do espalhamento da informação provê alta tolerância a falhas e auto-estabilização [8]. Alguns sistemas distribuídos utilizam esse protocolo para garantir que o dado seja disseminado para todos os nós de uma rede. Protocolos de comunicação baseados em boatos geralmente não requerem mecanismos de recuperação de falhas, o que os tornam muito simples.

Por sua natureza, este protocolo é utilizado para agregar e computar valores da rede, e valores agregados são geralmente mais importantes que valores locais. A abordagem do aprendizado via boatos envolve percorrer randomicamente a rede P2P, e toda vez que um nó é visitado, o valor agregado é atualizado utilizando um registro local.

Em [8] é exposto um algorítmo para a computação da média dos valores da rede:

- Cada nó inicia com a soma s_i e peso w_i;
- Inicialmente s_i é seu próprio registro e $w_i = 1$;
- Em cada *round*, o nó envia $s_i/2$ e $w_i/2$ para um vizinho randômico e armazena como seus próprios respectivos valores $s_i e w_i$;
- Em cada *round* metade do seu valor é enviado para alguém e metade é salvo;

A informação é difundida uniformemente pela rede até chegar na estabilidade. O quão rápida a informação se difunde pela rede depende do tipo de arquitetura da rede. Em determinados cenários de arquiteturas descentralizadas, a difusão é bastante rápida [8].

Assim, o protocolo de propagação por boatos tem bastante aderência na computação de agregações numéricas, em uma arquitetura descentralizada, e com boa tolerância a falhas.

Na próxima sessão, é exposto como o protocolo de propagação por boatos pode ser utilizado para implementar um algoritmo distribuído utilizando análise de componentes principais.

3.4.3 Abordagem Distribuída por Boatos

Nesta sessão iremos mostrar uma abordagem distribuída utilizando o protocolo de propagação de boatos em um modelo de dados de amostras distribuídas. De maneira geral, consideramos um conjunto N de observações em T possíveis agentes:

$$X = (x(1), x(2), ..., x(T)) \in \mathbb{R}^{N,T}$$

Para o modelo de amostras distribuídas, particionamos X por suas colunas:

$$X = (X_1^C, X_2^C ... X_S^C), \text{ onde } X_i^C \in \mathbb{R}^{N,T_i}$$

X_i é a porção de dados salva pelo agente i e $T = \sum_{i=1}^{S} T_i$.
Para o modelo de coordenadas distribuídas, particionamos X por suas linhas:

$$X = ((X_1^r)^T, (X_2^r)^T, ..., (X_S^r)^T)^T, \text{ onde } X_i^r \in \mathbb{R}^{N_i,T}$$

A matriz de correlação da amostra XX^T e a soma das instâncias $X1$ são suficientes para computar PCA [4]. A matriz de correlação da amostra e a soma das instâncias podem ser calculadas utilizando computações parciais de cada nó. Considerando $\mu_i = \frac{1}{n_i} X_i 1$ e $B_i = X_i X_i^T$, de acordo com [4]:

$$\frac{1}{n} X1 = \frac{1}{n} \sum_{i=1}^{N} X_i 1 = \frac{1}{\sum_{i=1}^{T} n_i} \sum_{i=1}^{N} X_i 1$$

$$C = \frac{1}{n} XX^T - \mu\mu^T = \frac{1}{n} \sum_{i=1}^{N} X_i X_i^T$$

Sendo assim, tornamos o problema do PCA distribuído como um problema de cálculo de médias distribuído. Dessa forma, como exibido anteriormente, iremos utilizar o protocolo de propagações de boatos. O algoritmo a seguir, retirado integralmente de [4], ilustra o procedimento de emissão e recepção de dados, onde não há nenhuma sincronia entre nós e com os nós na rede em uma arquitetura *Mesh*:

```
#Envio de Dados
def send_data(i):
    a[i] = X1[i]
    G[i] = X[i]*(X[i].T)
    w[i] = n[i]
    #calcula autovalores e autovetores da matriz G[i]
    V[i], L[i] = eigendecompose(G[i])
    '''Calcula U[i], definido no capítulo anterior,
    usando a matriz de autovalores e os autovetores'''
    U[i] = X[i]*V[i]*pow(lambda[i],-0,5)
    while(True):
        #propagação por boatos, calculo de médias
        j = random_neighbor(i)
        a[i], L[i], w[i] = 0.5*a[i],0.5*L[i],0.5*w[i]
        Send(a[i],U[i],L[i],w[i]) to j
```

```
#Recebimento de Dados
def receive_data(i):
    while(True):
        when_receive(a[j],U[j],L[j],w[j])
        #incrementa valores locais
        a[i] = a[i]+a[j]
        w[i] = w[i]+w[j]
        Q0 = U[i]
        for t in range(0,MAX_T):
            '''utilizando a decomposição QR, para decompor em uma matriz ortogonal Q
            e uma matriz diagonal R e ir somando os valores intermediários de modo
            que na última iteração tenhamos U[i] e L[i]'''
            Q[t+1],R[t+1] = qr_decomposition(U[t]*(U[t].T)*Q[t]+U[j]*(U[j].T)*Q[j])

        #na última iteração temos U[i] e L[i] calculados!
        U[i] = Q[MAX_T]
        L[i] = diagonal(R[MAX_T])
```

Esta abordagem é uma implementação correta de PCA com dados distribuídos, utilizando propagação de boatos. Porém, tem um grande grande desvantagem que pode tornar a implementação inviável. Há o envio de matrizes de altas dimensões, sendo inviável elas serem transmitidas pela rede.

Dessa forma, reduzir o tamanho dessas matrizes é um ponto importante. Não tiramos proveito do fato do *rank* da matriz de covariância ser baixo, sendo que o número de observações de cada amostra é muito maior que o número de *features*, isto é $n_i >> D$, sendo D o número de *features* [4]. No fim, trocamos informações que serão descartadas após o processo de redução de dimensionalidade, já que a redução de dimensionalidade ocorre após o processo de convergência da troca das informações na rede [4].

Há diversos outro casos nesse problema que ainda podem ser considerados. Em [6] há uma abordagem em outros tipos de rede, como em uma arquitetura *master-slave*, além da análise do algoritmo para o caso de coordenadas distribuídas, abordada brevemente neste trabalho. Em [4], há um estudo do quão conexa é a rede, já que a convergência do protocolo de propagação por boatos depende da infraestrutura da rede. É mostrado que o protocolo de propagação de boatos é adequada para redes com muitos nós, amostras grandes, e é muito robusta a falha de nós e partições de rede.

Dentro do viéis desse trabalho, para trabalhos futuros, faz sentido utilizar uma redução de dimensionalidade antes do processo de troca das informações, de modo a tirar proveito do *rank* de C_i, e há bibliografia que trata desse tipo de caso, como [6] e [4].

3.5 Conclusão

Apresentamos um algoritmo para resolver PCA quando os dados possuem amostras distribuídas, em virtude deles não poderem serem centralizados, por inúmeras questões como privacidade, custo ou legalidade. Para isso, mostramos como o protocolo de propagação por boatos pode ser utilizado nesse tipo de problema. Exibimos de maneira geral questões sobre o modelo dos dados, modelo de comunicação e infraestrutura.

O problema de análise de componentes principais sob o ponto de vista dos modelos de dados tratados (amostras e coordenadas distribuídas) há literatura disponível e estudos bem encaminhados. Porém, para particionamento irregular dos dados, não atendendo essencialmente as duas condições, há pouca literatura sobre o assunto. Um outro ponto é a carência de estudos a respeito de como a solução PCA distribuída e centralizada se diverge, de modo a avaliar não somente a convergência do algoritmo, mas também a corretude em relação a implementação usual.

A área de aprendizado distribuído têm evoluído substancialmente nos últimos anos. A *Google* em 2017 lançou uma infraestrutura, como serviço, para o aprendizado em base de dados distribuídas chamada *Federated Learning* [9], possibilitando que telefones celulares possam colaborativamente aprender um modelo compartilhado de predição, enquanto mantém os dados de treino no dispositivo, desacoplando a necessidade de armazenar os dados na nuvem.

A abordagem utilizada neste trabalho para atacar o problema de amostras distribuídas é inviável se os dispositivos possuem uma grande quantidade de dados. Há estudos com outras técnicas que implementam algoritmos mais eficientes em relação ao uso de memória. Em [4] há o estudo sobre modificações no algoritmo proposto para diminuir a quantidade dos dados trocados nas mensagens, ao aplicar redução de dimensionalidade antes do processo de comunicação entre os *peers*.

Novas demandas para o uso de algoritmos com amostras distribuídas tende a crescer com a preocupação a respeito da privacidade dos dados, com novas leis surgindo assegurando ao usuário uma segurança maior sobre o uso de suas informações pessoais.

Referências

[1] KONEčNý, J.; MCMAHAN, H. B.; RAMAGE, D.; RICHTáRIK, P. Federated optimization: Distributed machine learning for on-device intelligence, 2016.

[2] Lgpd, Jul 2021. Available at `https://www.lgpdbrasil.com.br/`.

[3] SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding machine learning - from theory to algorithms*. Cambridge University Press, 2014.

[4] FELLUS, J.; PICARD, D.; GOSSELIN, P.-H. Asynchronous gossip principal components analysis. *Neurocomputing*, v. 169, p. 262–271, 2015. Learning for Visual Semantic Understanding in Big Data ESANN 2014 Industrial Data Processing and Analysis.

[5] JOLLIFFE, I. *Principal component analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 1094–1096.

[6] WU, S. X.; WAI, H.-T.; LI, L.; SCAGLIONE, A. A review of distributed algorithms for principal component analysis. *Proceedings of the IEEE*, v. 106, n. 8, p. 1321–1340, 2018.

[7] ORMáNDI, R.; HEGEDűS, I.; JELASITY, M. Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience*, v. 25, n. 4, p. 556–571, May 2012.

[8] KEMPE, D.; DOBRA, A.; GEHRKE, J. Gossip-based computation of aggregate information. In: . c2003. p. 482–491.

[9] Federated learning: Collaborative machine learning without centralized training data, Apr 2017. Available at `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`.

4 Algoritmos para Mitigacão de latência em First Person Schooter Games Distribuídos

por Pedro Magalhães

4.1 Contexto

First Pearson Shooter (FPS), é um estilo de jogo de tiro feito para computadores e video games em que o jogador tem a visão em primeira pessoa focado em combate. Nesse capitulo vamos focar nos FPS competitivos, em que jogadores jogam uns contra os outros em um ambiente distribuído.

Por causa desse ambiente distribuído, para garantir um ambiente mais justo para os jogadores, os jogos precisam ter uma preocupação a mais em como tratam a latência de rede de cada usuário. Quando um jogador vê outro jogador em movimento, mira e dispara, surge um problema para o jogo para definir se aquele disparo vai acertar ou não o jogador.

Esse problema ocorre porque é complicado definir qual é o posicionamento real de um jogador em um determinado momento do jogo. Quando um jogador dispara é possível que o outro jogador já tenha se movimentado em outra direção mas essa informação ainda não chegou. Se a latência dos dois jogadores for muito baixa, ou seja o tempo para que os jogadores enviem e recebam informações do servidor seja baixo, o problema de decisão é quase irrelevante porque o posicionamento da tela do outro jogador vai ser o real posicionamento.

No entanto, se a latência de um dos jogadores é muito alta no cenário do disparo, se torna complicado decidir se o tiro aceta ou não. Para o jogador que atira, pode ser frustrante mirar perfeitamente em outro jogador e seus disparos nunca o acertarem. Enquanto para quem é atingido pode ocorrer de o disparo ser registrado quando ele já está em um local em que o outro jogador se quer deveria vê-lo.

4.2 Área de aplicação

Nos últimos anos a industria de jogos tem visto um grande crescimento. Segundo uma reportagem de 2019 da BBC[1], o valor dessa industria já superava a de filmes e música somados. Somente uma franquia de jogos FPS chamada *Call of Duty*, faturou no ano passado um bilhão e novecentos milhões de dólares[2].

O *Call of Duty*, por exemplo, possui um jogo chamado *Warzone* em que 150 pessoas podem jogar simultaneamente em uma mesmo mapa. Nesses jogos competitivos de FPS, em geral, temos uma arquitetura cliente servidor. O jogador renderiza o jogo em seu computador ou video game que guarda as informações referentes ao seu personagem e fica constantemente atualizando suas informações com o servidor do jogo, nessas atualizações ele pega também informações das ações de outros jogadores. Apesar do cliente lidar com as informações do jogador, o servidor é quem decide o real posicionamento dele, por exemplo, se a latência aumenta subitamente para um jogador, provavelmente o servidor vai corrigir a posição correta desse jogador quando conseguirem se comunicar. O jogador permanece conectado ao servidor durante a partida, em caso de desconexão o jogador é eliminado dessa partida.

Em geral os jogos tentam agrupar os jogadores de acordo com a região em que eles estão, principalmente nos jogos que contam com uma quantidade grande de jogadores. Além disso, os jogos tentam colocar servidores espalhados pelo mundo para que . Isso tudo é feito para que a latência entre o jogador e o servidor seja pequena e também, para que a diferença de latência entre os jogadores seja minimizada. Nos EUA, por exemplo é muito comum que exitam servidores na costa leste, na costa oeste e na região central.

4.3 Principais desafios sendo tratados

Nos jogos competitivos de FPS, uma diferença de latência de rede entre os jogadores pode ter um efeito de vantagem para um deles dependendo de como o servidor trata essa diferença. Isso porque, se um jogador, por exemplo, vê o outro primeiro, ele vai reagir primeiro tendo a vantagem em um confronto. Isso degrada a experiência do usuário, uma vez que seu desempenho vai ser diretamente impactado não só pela latência de sua conexão mas também pela conexão dos demais jogadores [3].

Em um jogo em que não há o tratamento da latência dos jogadores, quando um jogador dispara contra o outro ele precisa mirar na posição real desse jogador e não na posição que eles estão vendo na tela, tentando prever onde o outro jogador vai estar.

Um exemplo de problema causado pela latência em jogos FPS é chamado de *shot behind cover*, ou tiro por trás de cobertura. Isso ocorre quando quando um jogador, "A", está sendo atingido por um inimigo, "B", e se protege atrás de alguma cobertura, no entanto devido à latência do jogador "A", para o servidor, o jogador "A"já havia perdido antes de se abrigar. Ou seja, para o servidor e para "B"o jogador "A"nunca chegou a ficar atrás da cobertura, mas para o jogador "A"ele perdeu a batalha quando já estava protegido causando frustração para o jogador. Esse problema geralmente é efeito colateral do tratamento de latência nos servidores, o primeiro algoritmo que vamos mostrar acaba causando esse problema.

Outro problema comum, que é informalmente conhecido como *Peeker's advantage*[4], ou vantagem de quem espia, ocorre quando um jogador, "A", sai de trás de uma cobertura e vê outro jogador, "B", "A"tem a vantagem por causa do tempo que demora para o servidor informar a "B"do novo posicionamento do jogador "A". Os algoritmos que veremos a seguir não tratam esse problema.

Para contornar os problemas envolvendo a latência da rede nesses jogos, alguns métodos de mitigação de latência podem ser utilizados. No entanto, o uso desses algoritmos deve ser bem avaliado pois impacta diretamente a experiência dos jogares. Nesse capitulo vamos dar uma ideia geral de é a arquitetura dos jogos FPS, mostrar melhor o impacto da latência de rede, comentando em seguida sobre alguns artigos relacionados. Discutiremos também sobre dois métodos de mitigação de latência, explicando brevemente seu funcionamento, suas aplicações e por fim faremos uma comparação entre eles.

O desafio que vamos tratar nesse capitulo é de como coordenar a interação entre dois jogadores com, potencialmente, grandes diferença de latência. Esse é um problema que pode atrapalhar muito a experiência dos jogadores. A figura 4 demonstra essa relação.

No já citado estudo de Mark Claypool e Kajal Claypool[3], eles mostram uma correlação entre o desempenho dos jogares e a latência. Pelo estudo, os jogos de FPS são um dos gêneros que tem o maior impacto no desempenho.

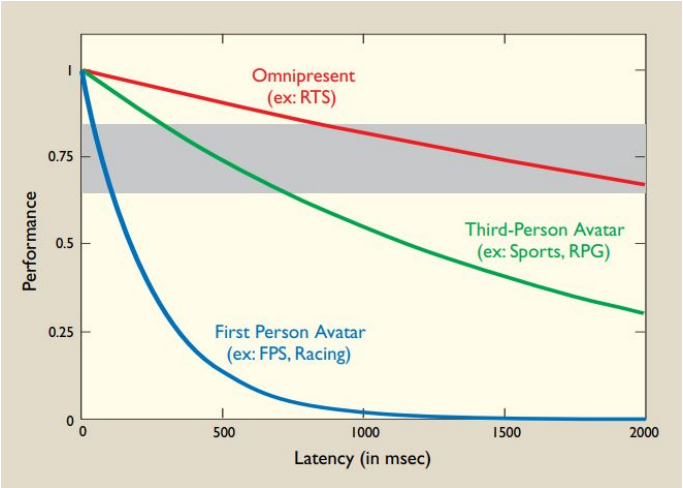


Figura 4: Relação entre a latência e o desempenho dos jogadores

4.4 Algoritmos da solução

Por causa desse impacto, a maioria dos jogos usa algum tipo de tratamento para melhorar o experiência do jogo, mesmo quando a latência está alta. Esse tratamento é chamado de algorítimos de mitigação de latência. Aqui vamos caracterizar dois desses algoritmos.

Para melhor demonstrar o problema a figura 5 mostra o que seria a posição real de um jogador adversário e a posição vista por outro jogador. O personagem renderizado é a posição real do jogador, enquanto os polígonos vermelhos e azuis representam a posição vista pelo outro jogador. Nesse cenário, se não tivermos um algoritmo de mitigação da latência e o jogador disparasse o tiro não seria registrado como acerto, mesmo que a mira esteja correta para o jogador que atirou. Assim, para que o jogador consiga acertar os tiros ele teria que prever onde o outro jogador realmente está.

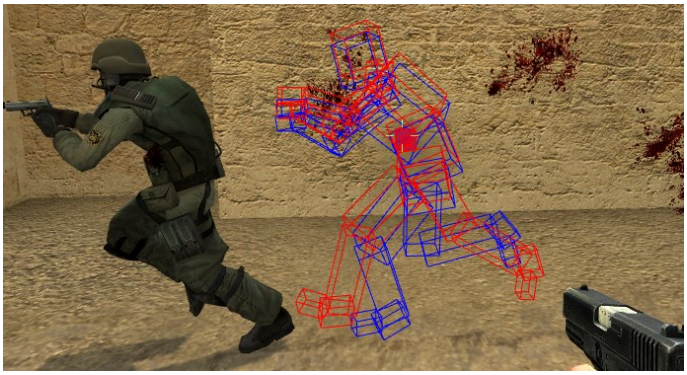


Figura 5: Posição real do jogador e posição vista por outro jogador

Na maioria dos jogos atuais, algum método de mitigação de latência é utilizado. O primeiro algoritmo que vamos mostrar, tenta manter um histórico de posicionamento dos jogadores para verificar se quando o tiro foi disparado o outro jogador estava na direção do projetil ou não. O outro algoritmo é uma modificação do anterior que tenta solucionar o problema do*shot behind cover* introduzido pela compensação da latência.

Para os dois algoritmos que vamos ver, eles tem que usar uma espécie de ordenação global de eventos dos jogadores. Ou seja, o servidor tem que conseguir definir se o disparo de um jogador ocorre antes ou depois da movimentação de outro jogador.

Para isso, o servidor e os clientes ficam em constante comunicação. Usualmente cada servidor sua frequência de atualização chamada de *tick rate*[5], um servidor com um *tick rate* de 20 significa que o servidor vai se comunicar com cada cliente 20 vezes por segundo. Nessas comunicações o servidor aproveita para comunicar o tempo atual do servidor, cada cliente mantém esse tempo atualizado e quando quer enviar algum comando para o servidor ele envia o último tempo visto antes do comando. Com essas informações o servidor consegue definir uma ordenação para as ações de cada jogador.

4.4.1 Algoritmo de Bernier

O primeiro algoritmo que vamos discutir é o de Bernier[6]. Esse algoritmo propõe que o servidor guarde o histórico do estado de cada jogador de tal forma que seja possível verificar a posição de cada um em um determinado momento. Assim, quando um jogador envia para o servidor um comando de disparo por exemplo, antes de avaliar o impacto do comando, o servidor verifica a posição dos outros jogadores quando o comando foi feito. Na imagem 5, os polígonos vermelhos seriam a posição que o servidor encontrou para o jogador e os azuis seriam a posição vista pelo outro jogador.

Para que esse algoritmo funcione o cliente e o servidor ficam sempre com um contador de *clock* atualizado. A cada atualização recebida do servidor o cliente atualiza esse valor. Assim, o servidor consegue saber em que momento um determinado comando foi feito por um jogador.

```
Before executing a player's current user command, the server:

      Computes a fairly accurate latency for the player

      Searches the server history (for the current player) for the world
      update that was sent to the player and received by the player just
      before the player would have issued the movement command

      From that update (and the one following it based on the exact target
      time being used), for each player in the update, move the other players
      backwards in time to exactly where they were when the current player's
      user command was created.  This moving backwards must account for both
      connection latency and the interpolation amount[8] the client was using
      that frame.

Allow the user command to execute (including any weapon firing commands,
etc., that will run ray casts against all of the other players in their "old"
positions).

Move all of the moved/time-warped players back to their correct/current
positions
```

Figura 6: Pseudo código Bernier

A lógica do servidor é descrita na figura 6 e funciona da seguinte forma. Antes de executar qualquer comando de um jogador o servidor faz o seguinte:

- Computa a latência do jogador
- Procura no histórico do servidor para esse jogador, a atualização enviada para ele logo antes do comando ser recebido.

- A partir dessa atualização move cada outro jogador para a posição que eles estavam, levando em conta a latência do jogador.
- permite que o comando recebido seja executado, se for um disparo a verificação de acerto vai levar em conta a posição calculada acima.
- por fim retorna todos os jogadores que "voltaram no tempo"para a posições reais

Com a utilização dessa compensação de latência conseguimos resolver o problema do jogador que dispara. No entanto ele introduz o problema do *shot behind cover*, em que que está sendo atingido pode já nem estar na linha de visão do outro jogador que atirou. Isso em geral ocorre quando jogadores com muita latência estão disparando. Em caso de latência muito alta, alguns jogos podem desabilitar o algoritmo de mitigação de latência para esses jogadores, tentando evitar o problema.

4.4.2 Compensação avançada

Nesse algoritmo[7] o autor propõe uma melhoria em cima do algoritmo anterior, tentado resolver o problema gerado pelo anterior. Introduzindo um novo passo de confirmação de acerto. É implementado no cliente uma forma de detectar que ocorreu um *shot behind cover* e nesse casos o cliente informa o servidor que não foi acertado.

Agora o acerto de um disparo só é confirmado quando o servidor recebe a confirmação da vitima. O servidor ainda faz o mesmo procedimento de verificar o estado do mundo na hora do disparo de um jogador e avaliar se o disparo é um acerto ou não. No entanto, agora quando ele verifica um acerto o servidor envia para vitima uma mensagem de acerto e aguarda a confirmação.

Esse algoritmo é feito para proteger jogadores com uma latência baixa, para jogadores com alta latência a etapa adicional de confirmação não é feita. Seguindo o autor, esse escolha foi feita para que não fosse injusto com os jogadores de baixa latência, já que esses jogadores são menos propensos a causar um caso de *shot behind cover*.

O algoritmo define também a possibilidade do servidor negar quando uma vitima informa que não foi atingida. Isso é feito para garantir que não exista a possibilidade de trapaça ou algum erro de calculo no cliente. Isso fica descrito na imagem a direita da figura 7.

Com esse algoritmo não há a necessidade desabilitar o algoritmo de compensação de latência para jogadores com o a latência mais alta. Esses jogadores apenas não vão ter a chance de "recorrer"de acertos. O principal impacto negativo do algoritmo é que ele adiciona um tempo adicional para a confirmação do acerto.

4.5 Conclusão

O problema da latência dos jogadores em jogos competitivos de FPS é bem complicado porque, por um lado, os jogos querem que o máximo de pessoas joguem o seu jogo, então pessoas com alta latência tem que ter uma experiência similar as pessoas com uma baixa latência. O uso de algum tipo de algoritmo de mitigação de latência nos parece essencial.

Vimos dois exemplos nesse capitulo, nos dois casos alguns efeitos colaterais acabam acontecendo. No caso do primeiro, jogadores com uma latência suficientemente alta podem acertar disparos em um outro jogador que ele se quer deveria estar vendo.

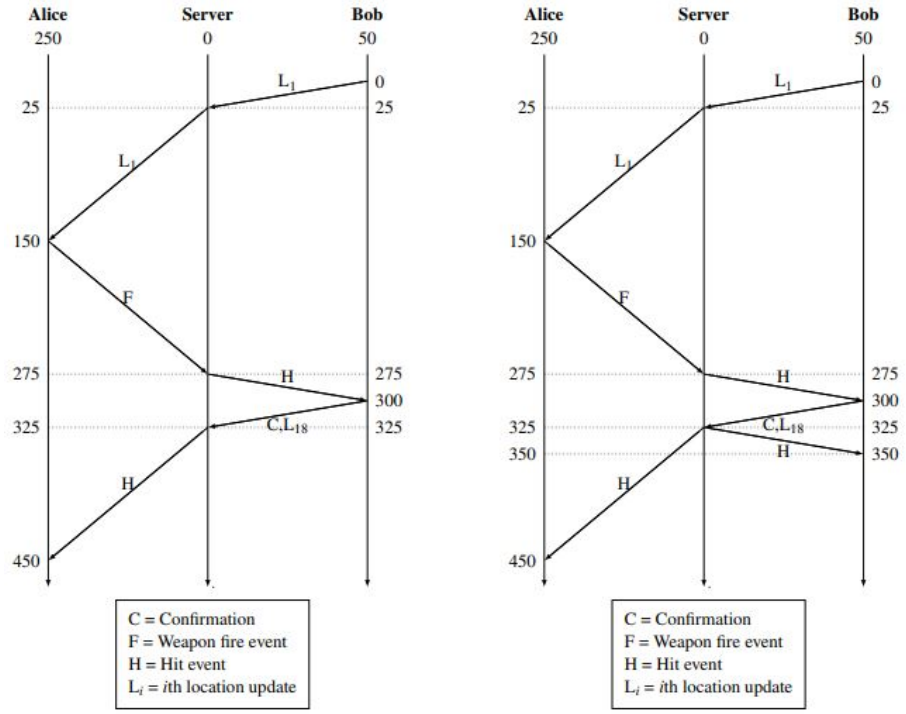


Figura 7: Algoritmo de confirmação de acerto

Já no caso do segundo vimos que há uma vantagem para os jogadores com baixa latência por que apenas esses tem uma proteção contra o *shot behind cover*, apesar dos autores afirmarem que é menos provável que um jogador com baixa latência cause um *shot behind cover*. Nos dois algoritmos é resolvido o problema de o jogador ter que prever a posição do outro jogador e ter que atirar "no ponto futuro"para conseguir acertar seus disparos.

Segundo Bernier, a maioria dos jogos de FPS implementa a compensação de latência porque os prós superam os contras causados pela uso da compensação.

Devido aos altos valores movimentado por jogos hoje em dia, muita pesquisa deve ser feita nessa área que não fica disponível, porque pode significar uma vantagem competitiva para as franquias de jogos. É muito difícil ver o código fonte dos jogos por exemplo. Só conseguimos ver dos jogos mais antigos, como o *Half Life* dos anos 2000 que apresenta o primeiro algoritmo discutido aqui.

Referências

[1] Gaming worth more than video and music combined, Jan 2019. Available at `https://www.bbc.com/news/technology-46746593`.

[2] BHAT, K. Modern warfare & warzone was the highest earning premium game of 2020, Jan 2021. Available at `https://charlieintel.com/modern-warfare-warzone-was-the-highest-earning-premium-game-of-2020/76106/`.

[3] CLAYPOOL, M.; CLAYPOOL, K. Latency and player actions in online games. *Commun. ACM*, New York, NY, USA, v. 49, n. 11, p. 40–45, Nov. 2006.

[4] STAVROPOULOS, A. Peeker's advantage in valorant explained, Mar 2021. Available at `https://dotesports.com/valorant/news/peekers-advantage-in-valorant-explained`.

[5] LEE, W.-K.; CHANG, R. K. Evaluation of lag-related configurations in first-person shooter games. In: . c2015. p. 1–3.

[6] BERNIER, Y. W. Latency compensating methods in client/server in-game protocol design and optimization. In: . c2001. v. 98033.

[7] LEE, S. W. K.; CHANG, R. K. C. Enhancing the experience of multiplayer shooter games via advanced lag compensation. In: . MMSys '18. New York, NY, USA: Association for Computing Machinery, c2018. p. 284–293.

5 Algoritmos de Consenso em Blockchain para Registros Invioláveis de Ações de Recuperação Ambiental

por Fernando Lima

Essa seção descreve um projeto de prova de conceito utilizando blockchain e uma forma personalizada de consenso para aplicações que pontuem e deem incentivos a ações de recuperação ambiental. A tecnologia Blockchain é frequentemente utilizado como uma fonte segura para registros de documentação e transações financeiras, funciona como alicerce de aplicações como cadeias de transções para registro de crédito de carbono [1] [2] [3], bem assim como transações de crédito para energia [4] e outros recursos gerados.

O projeto propõe a utilização de uma cadeia (blockchain) para efetuar registros invioláveis de ações de recuperação ambiental feita por instituições ou empresas em troca de créditos fiscais. Reflorestamento, despoluição, revitalização de parques, saneamento de lagoas e praias e outras medidas de recuperação ambiental são exemplos a ser considerados como ações ambientais a serem incentivados aravés de créditos fiscais.

A simulação do Blockhain foi implementada no software Sinalgo [5], capaz de simular nós conectados em rede que enviam mensagens ao longo do tempo, podendo conter modelos de mobilidade e interferência de conexão. Na Simulação específica do projeto, os agentes representados pelos nós do Sinalgo, são instituições conectadas ao blockchain, permitindo a inserção de blocos a cadeia por parte de todos.

5.1 Sistema de Créditos

Cada ação de restauração ambiental deve ser recompensada com um valor em cripto. Perdas ou desistências podem acarretar em perdas da cripto. Ao alcançar N_t nível de suporte, no qual será detalhado adiante, uma instituição pode adicionar um bloco. O nível de suporte é uma proposta de esforço na qual uma instituição deve cumprir para inserir um bloco a cadeia, colecionando um determinado patamar de nível de suporte. A contagem do nível é incrementada por um ou mais agentes reguladores que validam as propostas e execução de ações ambientais das instituições. O conceito de mérito ou esforço computacional foi um das primeiros métodos de consenso e validação em Blockchains, onde a adição de um bloco através da realização desse esforço é financeiramente recompensada. No caso do Blockchain Ambiental aqui apresentado, o esforço e a recuperação ambiental são simultâneos, logo um membro é diretamente recompensado pela ação. A tabela 4 mostra exemplos de possíveis ações ambientais, cada ação é contribui como um esforço para inserir um bloco na cadeia e está vinculada a uma recompensa.

Tabela 4: Exemplos de tipos de suporte

Ação ambiental	Recompensa	Suporte
Reflorestamento	C$ 3,00 criptos por custódia;	+1 nível de suporte;
Despoluição	C$ 1,00 criptos por custódia;	+1 nível de suporte;
Limpeza de Afluentes	C$ 2,00 criptos por custódia;	+1 nível de suporte;

5.2 Blockchain

Blockchain [6] é um tipo de banco de dados descentralizado onde cada registro é imutável. Uma diferença importante entre um banco de dados típico e um blockchain é a maneira como os dados são estruturados. Uma blockchain coleta informações em grupos, também conhecidos como blocos, que contêm conjuntos de informações. Os blocos têm certas capacidades de armazenamento e, quando preenchidos, são encadeados no bloco previamente preenchido, formando uma cadeia. Todas as novas informações que seguem aquele bloco recém-adicionado são compiladas em um bloco recém-formado que também será adicionado à cadeia depois de preenchido. As informações contidas em um bloco formam um hash, o que garante que as informações desse bloco de dados não foram violadas. Todo bloco criado contém sua hash e a do bloco anterior, criando uma conexão entre os blocos.

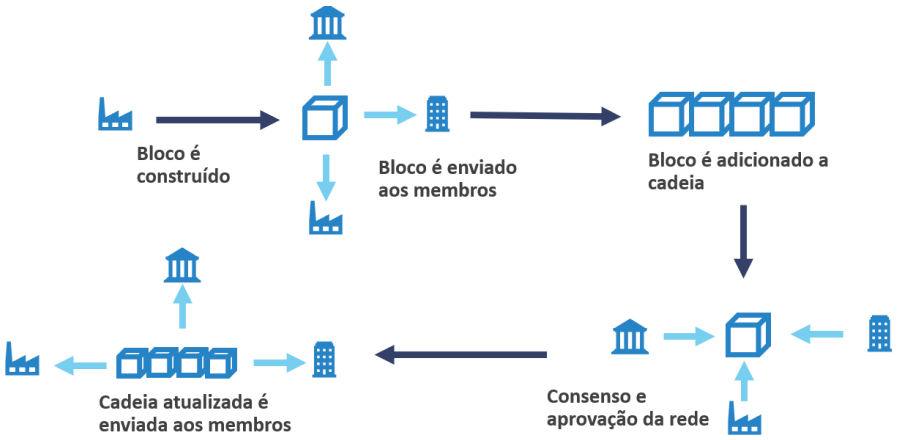


Figura 8: Os membros presentes no sistema distribuído possuem uma cópia do Blockchain, quando um membro passa pelo processo de criação do bloco, esse pode ser enviado para outros membros que podem adicionar o bloco a sua cópia da cadeia. Os membros passam por um processo de consenso para que uma das cadeias prevaleça sobre as demais, e as cadeias predominantes são distribuídas entre os membros, que atualizam o estado atual do sistema. Todo o processo ocorre simultaneamente

Os sistemas Blockchain são descentralizados e autorregulados e funcionam em escala global sem nenhuma autoridade central coordenadora. Eles envolvem contribuições de centenas de milhares de participantes que trabalham na verificação e autenticação de transações que ocorrem no Blockchain e nas atividades de criação (ou mineração) dos blocos. Os registros compartilhados publicamente precisam de um mecanismo eficiente, justo, em tempo real, funcional, confiável e seguro para garantir que todas as transações que ocorrem na rede sejam genuínas e todos os participantes concordem em um consenso sobre o status do sistema. Essa importante tarefa é realizada pelo mecanismo de consenso, que é um conjunto de regras que decide sobre as contribuições dos vários participantes da Blockchain.

5.3 Consenso

O protocolo de consenso [7] Blockchain consiste em alguns objetivos específicos, como chegar a um acordo, colaboração, cooperação, direitos iguais para todos os nós e partici-

pação obrigatória de cada nó no processo de consenso. Assim, um algoritmo de consenso visa encontrar um acordo comum que seja uma vitória para toda a rede. O processo de ganho de nível de suporte para alcançar o nível alvo, mostra-se como uma forma de esforço para adicionar um bloco a cadeia, esse processo será chamado de **Proof-of-Sponsorship (PoSp)**. Existem outras estratégias para inserção de blocos que não envolvem uma prova de esforço em si, mas outras formas de validação como a atribuição a autoridades escolhidas [8] ou sobre uma taxa atribuída a transação [9]

Os membros estão constantemente na tentativa de reunir níveis de suporte para cumprir o esforço, para que seus blocos sejam adicionados a suas cópias da cadeia e transmitido para outros membros adicionarem este bloco as suas cadeias, conforme a rotina ilustrada na figura 10. Nessa altura é notório a conclusão que membro podem ter cadeias distintas, ao longo do tempo uma cadeia deve prevalecer pelo consenso. Nesta parte é definido o conceito de que **a maior cadeia prevalece**. Os membros enviam periodicamente uma requisição de consenso, na qual é perguntada para outros membros o tamanho (em número de blocos) das suas cópias da cadeia. A cópia que tiver o menor tamanho será sobrescrita pela de maior tamanho enviada como resposta dos outros membros que respondem ao nó que requisitou o consenso.

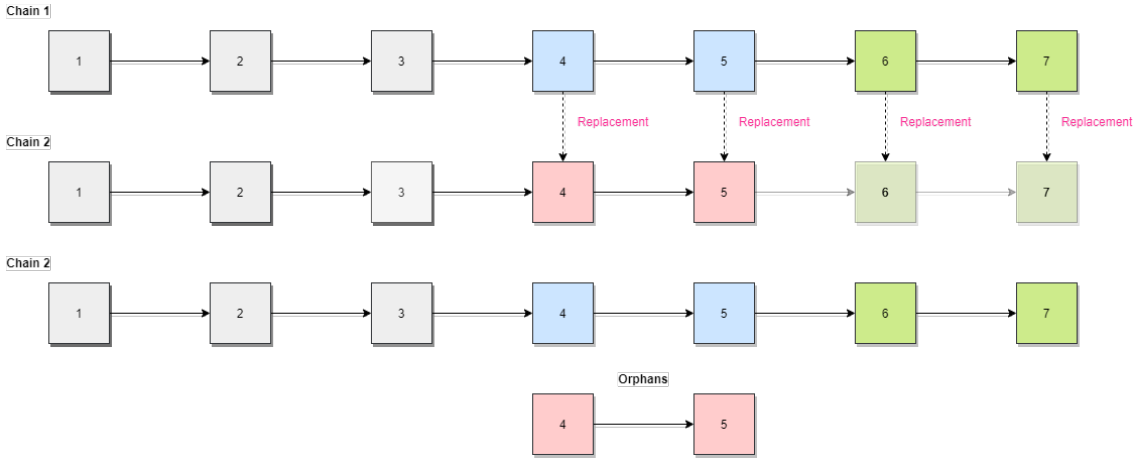


Figura 9: A cadeia 1 de tamanho 7 substitui a cadeia 2 durante o processo de consenso, os blocos divergentes (4 e 5) são descartados e ficam órfãos de cadeia.

A figura 9 descreve o processo de mudança de cadeia. A maior cadeia prevalece e os blocos que diferem da cadeia vencedora do processo de consenso são retirados da cadeia, estes blocos são chamados de órfãos e representam uma perda de informação. A estratégia para lidar com essa perda varia com a aplicação do Blockchain, esse projeto não entra nesse mérito, mas o mais comum é se realizar uma estratégia de recuperação, para reinserir a informação dos blocos órfãos em blocos posteriores.

5.3.1 Bloco e Operações

O bloco é composto pelas informações de identificação do bloco, o número do bloco, timestamp, o tipo e valor da transação. Todos os blocos armazenam o hash das suas informações e o hash do bloco anterior. Dessa forma é possível criar um encadeamento entre os blocos, se houver falha de segurança e a informação de um bloco for alterada,

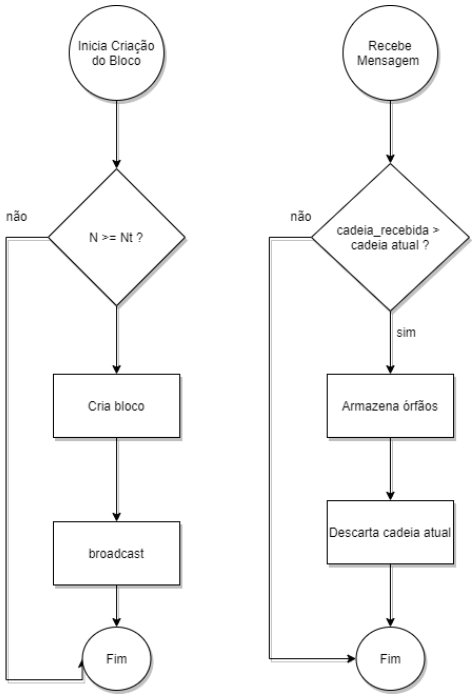


Figura 10: Rotina de envio do bloco e consenso

o hash irá ser alterado também, essa alteração pode ser checada pelo próximo bloco, ao comparar o hash armazenado do anterior que foi modificado, acusando que a cadeia foi violada daquele ponto em diante.

$$encriptedString_k = Hash(block_k)$$

$$previousHash_k = Hash(block_{k-1})$$

Cada bloco registra as transações entre os membros e as recompensas pelas ações ambientais. A informação de transação é registrada no bloco a partir de operações estabelecidas no sistema. As operações disponíveis são definidas pela tabela:

Tabela 5: Tipos de operação registradas nos blocos

Tipo	**Definição**	**Cifra**
Adição	Adição de cripto ao cumprir o PoSp	(@I1++$);
Subtração	Retirada de custódia de ação ambiental	(@I1–$)
Transferência	Transferência de custódia	(@I1->$@I2)

Conforme especificado na seção 5.1, cada ação é contribui como um esforço para inserir um bloco na cadeia e está vinculada a uma recompensa. As recompensas são acumuladas por instituição membro do Blockchain, assim como o nível de suporte. Quando o nível de suporte atinge uma constante global definida como limiar de suporte N_t, o esforço é recompensas com os valores das recompensas acumuladas em cripto-moedas e o bloco é formado, permitindo-se adiciona-lo a cadeia, conforme ilustrado na figura 11.

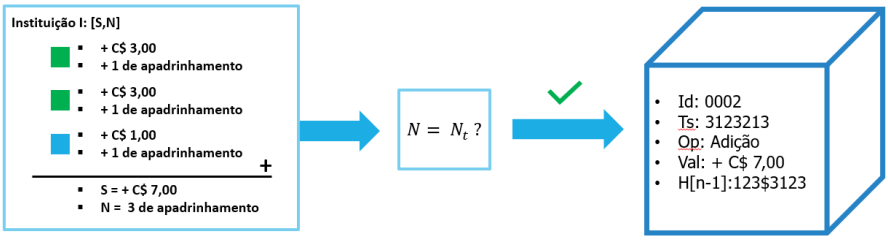


Figura 11: A instituição I se comprometeu a reflorestar duas áreas castigadas pelo desmatamento e limpar afluentes, logo será recompensada em um total acumulado de C$ 7,00 cripto-moedas e seu nível de suporte atual será de dois, ao atingir $N_t \geq 3$ o bloco é formado com as informações da instituição a operação (de adição) e o valor acumulado.

Anualmente, cada instituição poderá converter seus créditos em cripto em descontos fiscais nas alíquotas, baseado em cálculos a definir, no próximo ano fiscal. O saldo s de cada instituição k precisa ser maior a meta t calculada a partir das instituições. Todas as ações são listadas e precificados. A soma dos preços forma o número finito de criptos disponível. Novas ações podem ser adicionadas ou removidas do "mercado".

$$Desconto_k = f(s(I_k) - t(\mathbf{I})$$

5.4 Sinalgo

Sinalgo é uma ferramenta de simulação de algoritmos distribuídos que se presta à prototipação, teste e depuração de algoritmos distribuídos. Para tal, o simulador abstrai das camadas subjacentes de rede e enlace, bem como das características e quantidade dos recursos computacionais e de armazenamento dos nós envolvidos. Permite uma execução em modo síncrono (em rodadas) e em modo assíncrono, que consiste do escalonamento de eventos para momentos futuros da simulação. O seu modelo de programação oferece ao usuário uma visão de nós que interagem pela mera troca de mensagem e não compartilham qualquer recurso. O Sinalgo foi projetado para, mas não se limita a, simular nós móveis que interagem através de redes sem fio com uma cobertura indicata pelo programador.

Para fins de análise do consenso no Blockchain foram simulados no Sinalgo o comportamento de 10 nós completamente interligados. Cada nó implementa o serviço de produção de blocos e possui sua cópia do Blockchain. O Sinalgo é capaz de calcular a passagem de tempo global em rounds. Em cada round o Runtime do Sinalgo executa as rotinas de cada nó na simulação. Foram observados o comportamento dos nós durante 10000 Rounds. No final do ensaio, foi possível extrair algumas estatísticas descritas a seguir. Todos os nós mandam mensagens em condições iguais de latência, sem interferência e obstruções durante toda a simulação, contando somente como diferença o tempo de processamento de mensagens.

Na simulação específica foi observado o comportamento dos nós com mobilidade. A mobilidade permite que quaisquer nó se movimente ao longo do espaço. Conforme a movimentação um nó pode se conectar ao outro, foi estabelecido um raio de alcance de conexão e todos os nós vizinhos dentro desse raio são conectados. Com esse comportamento, é possível observar que ao longo do espaço, os nós não podem ser conectar com todos disponíveis

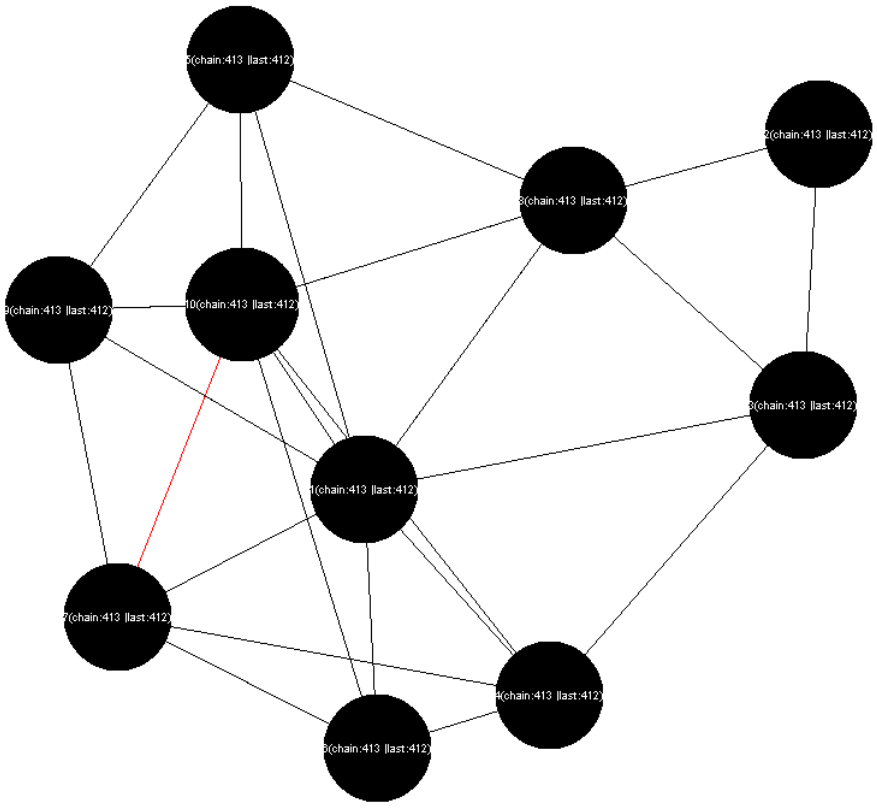


Figura 12: Simulação de 10 nós na plataforma Sinalgo. Cada Nó se movimenta ao longo do espaço, atualizando sua posição e conexão com outro nós.

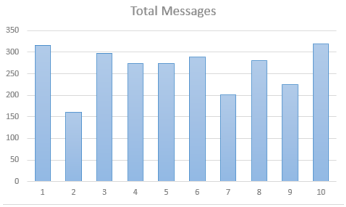
e formam-se disparidades nas cadeias dos nós, permitindo observar o comportamento do consenso distribuído.

5.4.1 Resultados

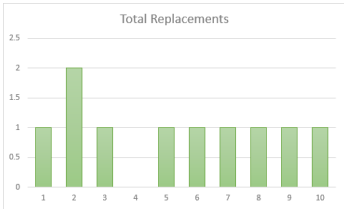
Na simulação foram estabelecidos três KPIs (Key Performance Indicators) para analisar o desempenho do sistema distribuído. Os indicadores foram escolhidos com a finalidade de analisar a perda de informação na substituição de cadeias. Foram escolhidos três KPIs

- **Total de Blocos Órfãos**: O número de blocos perdidos em uma troca de cadeia pela maior;
- **Número de trocas efetuadas**: A quantidade de trocas de cadeias efetuadas;
- **Número de mensagens efetuadas**: A quantidade total de mensagens;

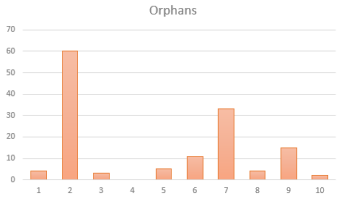
Os resultados apresentados nos histogramas na Figura 13 mostram uma troca de mensagens constante ao longo dos nós. Houveram poucas trocas de cadeia entre os membros, o que resultou em poucos órfãos por nó. Em contraste com o resto, temos o nó 2 no qual trocou poucas mensagens e teve mais trocas de cadeia, resultando em um número de órfãos



(a) Número total de mensagens por nó



(b) Número total de trocas de cadeia por nó



(c) Número total de órfãos por nó

Figura 13: Resultados dos KPIs

maior que a média entre nós. Esta anomalia pode ser explicada pela mobilidade do nó 2 que ficou mais distante e menos conectado aos outros nós, recebendo menos mensagem, necessitando de mais atualizações da cadeia.

5.5 Conclusão

Esse projeto apresenta uma implementação de um algoritmo distribuído baseado em consenso. O blockchain é a estrutura de dados distribuída mais atual e utilizada recentemente. O objetivo é alcançar o patamar de uma plataforma segura, descentralizada e automatizada dentro das possibilidades e questões legais. A estrutura de um "livro"de registros distribuído é considerado parte integral do futuro [10] para o sistemas financeiros e registros de documentação.

Posteriormente, propõe-se a evolução do procedimento utilizado para cumprir a inserção de blocos a cadeia. Com a mesma ideia apresentada em [11] projeta-se a combinação de de Proof-of-Sp para membros e Proof-of-Stake (ou Authority) para agentes validadores de propostas para a obtenção e incremento do nível de suporte, visando uma automatização do processo, evitando critérios subjetivos, fraudes e corrupções, além do aprimoramento de perdas de blocos órfãos. Espera-se estressar o sistema com simulações envolvendo uma mobilidade maior e interferência de envios de mensagem.

Referências

[1] BAI, Y.; SONG, T.; YANG, Y.; BOCHENG, O.; LIANG, S. Construction of carbon trading platform using sovereignty blockchain. In: . c2020. p. 149–152.

[2] HUA, W.; SUN, H. A blockchain-based peer-to-peer trading scheme coupling energy and carbon markets. In: . c2019. p. 1–6.

[3] MOSS.EARTH. Our greatest innovation: putting carbon credits into blockchain. "https://moss.earth/en/home/#solution".

[4] BAIG, M. J. A.; IQBAL, M. T.; JAMIL, M.; KHAN, J. Iot and blockchain based peer to peer energy trading pilot platform. In: . c2020. p. 0402–0406.

[5] ZURICH, D. C. G. E. Sinalgo - simulator for network algorithms. "https://sinalgo.github.io/index.html".

[6] BITCOIN. Bitcoin: A peer-to-peer electronic cash system. "https://bitcoin.org/bitcoin.pdf".

[7] JUAN A. GARAY TEXAS, AGGELOS KIAYIAS, N. L. The bitcoin backbone protocol: Analysis and applications. "https://eprint.iacr.org/2014/765.pdf".

[8] EKPARINYA, P.; GRAMOLI, V.; JOURJON, G. The Attack of the Clones Against Proof-of-Authority. In: . San Diego, CA: Internet Society, c2020.

[9] NAIR, P. R.; DORAI, D. R. Evaluation of performance and security of proof of work and proof of stake using blockchain. In: . c2021. p. 279–283.

[10] HUIBERS, F. Distributed ledger technology and the future of money and banking. *Accounting, Economics, and Law: A Convivium*, p. 20190095, 2021.

[11] DUONG, T.; FAN, L.; KATZ, J.; THAI, P.; ZHOU, H.-S. 2-hop blockchain: Combining proof-of-work and proof-of-stake securely. In: . Editors CHEN, L.; LI, N.; LIANG, K.; SCHNEIDER, S. Cham: Springer International Publishing, c2020. p. 697–712.