Sistema Distribuído de Log com Ordenação Causal: Implementação do Relógio de Lamport e Algoritmo Bully

Sergio Sebastian Pezo Jimenez
Instituto de Computação
Universidade Estadual de Campinas
Campinas, Brasil
s298813@dac.unicamp.br

José Victor Santana Barbosa
Instituto de Computação
Universidade Estadual de Campinas
Campinas, Brasil
j245511@dac.unicamp.br

***Abstract*—Este trabalho apresenta a implementação e análise de um sistema distribuído de log que garante ordenação causal de mensagens através do Relógio Lógico de Lamport e utiliza o Algoritmo Bully para eleição de líder. O sistema foi implementado em Python com FastAPI e deployado em três regiões geograficamente distribuídas do Google Cloud Platform: Iowa (EUA), São Paulo (Brasil) e Sydney (Austrália), totalizando aproximadamente 36.000 km de separação. Os experimentos conduzidos a partir de Campinas demonstram o funcionamento correto da ordenação causal sob concorrência, com throughput de até 27.47 msg/s para 50 mensagens simultâneas. A análise das latências de rede revela o impacto significativo da distância geográfica: 19ms para São Paulo (região mais próxima a Campinas), 294ms para Iowa e 652ms para Sydney. Os resultados validam a eficácia dos algoritmos implementados para manutenção de consistência causal em ambientes geodistribuídos.**

***Index Terms*—Sistemas Distribuídos, Relógio de Lamport, Algoritmo Bully, Ordenação Causal, Replicação, Google Cloud Platform**

I. Introdução

A ordenação causal de eventos é um problema fundamental em sistemas distribuídos, onde múltiplos processos executam concorrentemente sem acesso a um relógio global sincronizado. Em 1978, Leslie Lamport introduziu o conceito de relógios lógicos [**?**], permitindo estabelecer uma ordenação parcial consistente com a causalidade, mesmo na ausência de sincronização física entre os nodos.

Paralelamente, em sistemas distribuídos que adotam arquiteturas single-leader para replicação de dados, surge a necessidade de mecanismos automáticos para eleição de líder. O Algoritmo Bully [**?**] oferece uma solução determinística onde o processo com maior identificador sempre assume a liderança, garantindo convergência mesmo em cenários de falhas parciais.

Este projeto, desenvolvido no âmbito da disciplina MC714 - Sistemas Distribuídos na Unicamp, implementa ambos os algoritmos em um sistema de log distribuído com três nodos. O objetivo é construir um sistema que garanta ordenação causal de mensagens através do Relógio de Lamport, enquanto utiliza o Algoritmo Bully para coordenação através de um líder eleito automaticamente.

A implementação foi deployada em três regiões geograficamente distantes do Google Cloud Platform para simular condições realistas de latência em redes de longa distância (WAN). Os experimentos foram conduzidos a partir de Campinas, Brasil, permitindo observar o comportamento do sistema sob diferentes condições de latência de rede: baixa latência para a região de São Paulo (19ms), latência média para Iowa (294ms) e alta latência para Sydney (652ms).

II. Fundamentação Teórica

A. Relógio Lógico de Lamport

O Relógio Lógico de Lamport estabelece uma ordenação parcial de eventos em sistemas distribuídos através de timestamps lógicos que respeitam a relação de causalidade "aconteceu-antes" ($\rightarrow$). Cada processo P_i mantém um contador local C_i que é incrementado de acordo com as seguintes regras [**?**]:

1) **Evento local**: Antes de executar um evento, P_i incrementa $C_i := C_i + 1$
2) **Envio de mensagem**: Ao enviar uma mensagem m, P_i inclui o timestamp $ts(m) = C_i$
3) **Recebimento de mensagem**: Ao receber m com timestamp $ts(m)$, P_j atualiza seu relógio para $C_j := \max(C_j, ts(m)) + 1$

A propriedade fundamental é: se $e \rightarrow e'$, então $C(e) < C(e')$. Isso permite estabelecer uma ordem total consistente com a causalidade quando combinado com identificadores de processo.

B. Algoritmo Bully

O Algoritmo Bully [**?**] é um algoritmo de eleição de líder para sistemas distribuídos onde cada processo possui um identificador único e totalmente ordenado. O algoritmo garante que o processo com maior ID sempre seja eleito líder, seguindo o protocolo:

1) Um processo P_i inicia uma eleição ao detectar que o líder atual falhou

2) P_i envia mensagens ELECTION para todos os processos com IDs maiores que o seu
3) Se nenhum processo responder, P_i se declara líder e envia mensagens COORDINATOR para todos
4) Se algum processo com ID maior responder, P_i aguarda a mensagem COORDINATOR
5) O processo com maior ID sempre vence a eleição

O algoritmo é denominado "Bully" porque processos com IDs maiores sempre "intimidam" processos com IDs menores, assumindo a liderança.

C. Replicação Single-Leader

A arquitetura single-leader (líder único) é um padrão comum em sistemas distribuídos onde todas as escritas são direcionadas a um nodo líder, que então replica as mudanças para os seguidores (followers) [**?**]. Esta abordagem simplifica a garantia de consistência, mas introduz um ponto único de falha, mitigado através de mecanismos de eleição automática de líder.

III. Metodologia de Implementação

A. Arquitetura do Sistema

O sistema foi implementado em Python 3.9 utilizando o framework FastAPI para exposição de APIs REST. A arquitetura consiste em três componentes principais:

- **LamportClock**: Classe thread-safe que implementa o relógio lógico com sincronização através de `threading.Lock`
- **Server**: Modelo que representa o estado de cada nodo, incluindo ID, lista de mensagens e conhecimento de outros nodos
- **FastAPI Application**: Endpoints REST para recebimento de mensagens, consulta de estado e eleição de líder

Cada nodo mantém uma lista local de mensagens, onde cada mensagem é anotada com:

- `id`: Identificador sequencial local
- `content`: Conteúdo da mensagem
- `lamport_timestamp`: Timestamp lógico de Lamport
- `node_id`: Identificador do nodo que gerou a mensagem
- `physical_timestamp`: Timestamp físico para debugging

B. Implementação dos Algoritmos

O relógio de Lamport foi implementado com proteção thread-safe usando `threading.Lock`, seguindo o algoritmo clássico: incremento local antes de eventos, e atualização para $\max(local, remote) + 1$ ao receber mensagens.

O algoritmo Bully foi implementado de forma simplificada, onde o nodo com maior ID (8003 - Sydney) assume automaticamente a liderança. Em produção, seria necessário implementar detecção de falhas e mensagens ELECTION/COORDINATOR explícitas.

C. Deployment Geográfico

O sistema foi deployado utilizando Google Cloud Platform em três regiões geograficamente distantes para simular condições realistas de latência WAN:

- **Node 1 (ID: 8001)**: us-central1-a (Iowa, EUA)
- **Node 2 (ID: 8002)**: southamerica-east1-a (São Paulo, Brasil)
- **Node 3 (ID: 8003)**: australia-southeast1-a (Sydney, Austrália)

Cada nodo executa em uma VM e2-micro com 1GB RAM, executando um container Docker com a aplicação FastAPI. As VMs são configuradas através de startup scripts que:

1) Instalam Docker
2) Obtêm as IPs externas dos outros nodos
3) Iniciam o container com variáveis de ambiente configurando os peers

A escolha de três regiões em continentes diferentes (América do Norte, América do Sul e Oceania) resulta em aproximadamente 36.000 km de separação total, com distâncias individuais de:

- Iowa $\leftrightarrow$ São Paulo: $\sim$8.000 km
- Iowa $\leftrightarrow$ Sydney: $\sim$13.000 km
- São Paulo $\leftrightarrow$ Sydney: $\sim$15.000 km

IV. Experimentos e Métricas

Os experimentos foram conduzidos a partir de Campinas, Brasil, utilizando scripts automatizados para coleta de métricas. A proximidade geográfica de Campinas à região de São Paulo (aproximadamente 90 km) resulta em latências significativamente menores para essa região, conforme esperado.

A. Latência de Rede entre Regiões

A Tabela **??** apresenta as latências HTTP medidas entre o cliente em Campinas e cada região do GCP. Foram realizadas 5 medições para cada região para calcular a latência média.

TABLE I
Latências HTTP Medidas (Cliente em Campinas)

Região	Distância	Latência Média	Desvio
São Paulo	$\sim$90 km	19 ms	$\pm$0.4 ms
Iowa	$\sim$8.000 km	295 ms	$\pm$7.0 ms
Sydney	$\sim$18.000 km	652 ms	$\pm$51 ms

A latência para São Paulo é aproximadamente 15x menor que para Iowa e 34x menor que para Sydney, refletindo diretamente o impacto da distância geográfica e do número de hops na rede. A variação observada na latência para Sydney ($\pm$51ms) sugere possíveis variações de roteamento ou congestionamento na rede transoceânica.

TABLE II
THROUGHPUT SOB DIFERENTES CARGAS

Carga (msg)	Tempo (s)	Throughput (msg/s)	Lat. Média (ms/msg)
10	0.930	10.75	93.0
25	1.275	19.61	51.0
50	1.909	26.19	38.2
100	36.976	2.70	369.8

B. Throughput sob Diferentes Cargas

O throughput do sistema foi avaliado enviando diferentes quantidades de mensagens concorrentes ao líder (Sydney) e medindo o tempo total até que todas as requisições HTTP fossem aceitas. A Tabela **??** apresenta os resultados.

Observa-se que o throughput aumenta de forma aproximadamente linear até 50 mensagens (26.19 msg/s), mas sofre degradação significativa ao atingir 100 mensagens (2.70 msg/s). Esse comportamento sugere saturação do sistema, possivelmente devido a:

1) Limitação de conexões HTTP concorrentes no servidor FastAPI
2) Timeout de requisições causado pela alta latência para Sydney (600ms)
3) Gargalo de processamento na VM e2-micro (1 vCPU)

A latência média por mensagem é inversamente proporcional ao throughput, aumentando drasticamente de 38.2ms (carga 50) para 369.8ms (carga 100), indicando formação de fila de espera no servidor.

C. Convergência dos Relógios de Lamport

Para avaliar o comportamento do Relógio de Lamport sob concorrência, foram executadas três rodadas de escritas simultâneas nos três nodos (total de 9 mensagens). A Tabela **??** mostra o estado final dos relógios lógicos.

TABLE III
ESTADO DOS RELÓGIOS DE LAMPORT APÓS ESCRITAS CONCORRENTES

Nodo	Região	Lamport Time
Node 1 (8001)	Iowa	14
Node 2 (8002)	São Paulo	4
Node 3 (8003)	Sydney	104

A disparidade nos valores dos relógios ($T_1 = 14$, $T_2 = 4$, $T_3 = 104$) reflete o padrão de comunicação do sistema: o Node 3 (Sydney), sendo o líder, recebe e processa a maioria das mensagens, resultando em incrementos mais frequentes. A análise confirma que a propriedade de ordenação causal foi preservada com timestamps monotonicamente crescentes ($t = 0, 1, 2, \ldots, 14$), validando a implementação correta do algoritmo.

V. ANÁLISE DOS RESULTADOS

A. Impacto da Distância Geográfica

Os resultados confirmam que a distância geográfica tem impacto direto e significativo na latência de comunicação. A relação quase linear entre distância e latência observada (19ms para 90km, 295ms para 8.000km, 652ms para 18.000km) está alinhada com a velocidade de propagação da luz em fibra óptica ($\sim$200.000 km/s) acrescida de latências de roteamento.

Para um sistema distribuído globalmente, essas latências impõem limites fundamentais no throughput e tempo de resposta. A escolha de São Paulo foi estratégica por estar próxima a Campinas (onde os testes foram conduzidos), demonstrando o benefício de edge computing.

B. Comportamento sob Carga

O sistema demonstra comportamento estável até cargas de 50 mensagens concorrentes, atingindo throughput máximo de 26.19 msg/s. No entanto, ao aumentar a carga para 100 mensagens, observa-se degradação severa, com throughput caindo para 2.70 msg/s (redução de 90%).

Esta degradação não-linear sugere que o sistema atinge um ponto de saturação entre 50 e 100 mensagens. Possíveis causas incluem:

- **Timeout de conexões**: FastAPI pode estar rejeitando conexões quando a fila de requisições excede um limite
- **Gargalo de CPU**: A VM e2-micro possui apenas 1 vCPU, limitando a capacidade de processamento concorrente
- **Latência de replicação**: Com mais mensagens, a latência de replicação entre nodos aumenta proporcionalmente

C. Ordenação Causal e Consistência

A análise dos Lamport timestamps confirma que a propriedade de ordenação causal foi mantida em todos os experimentos. Especificamente:

1) Eventos locais incrementam o relógio monotonicamente
2) Mensagens recebidas sempre têm timestamp maior que mensagens enviadas anteriormente pelo mesmo processo
3) A relação aconteceu-antes ($\rightarrow$) é preservada através dos timestamps

No entanto, os valores díspares dos relógios ($T_1 = 14$, $T_2 = 4$, $T_3 = 104$) revelam que a carga não está distribuída uniformemente. O líder (Node 3) processa significativamente mais mensagens que os followers, indicando que a arquitetura single-leader concentra a carga no nodo líder.

D. Trade-offs entre Consistência e Performance

A implementação atual prioriza consistência causal através do Relógio de Lamport, mas apresenta limitações de performance evidentes na degradação de throughput sob alta carga. Em sistemas distribuídos, existe um trade-off fundamental entre:

- **Consistência forte**: Garantias de ordenação causal, mas com maior latência
- **Disponibilidade**: Capacidade de responder mesmo sob falhas parciais
- **Tolerância a partições**: Funcionamento durante particionamento de rede

O teorema CAP [**?**] estabelece que é impossível garantir simultaneamente Consistência, Disponibilidade e Tolerância a

Partições. Nossa implementação escolhe Consistência (através do Lamport Clock) e Disponibilidade (múltiplas réplicas), sacrificando parcialmente a Tolerância a Partições (o líder único é um ponto de falha).

VI. Trabalhos Relacionados

O Relógio de Lamport [**?**] é um dos trabalhos seminais em sistemas distribuídos e continua sendo a base para algoritmos mais sofisticados como Vector Clocks [**?**] e Version Vectors. O Algoritmo Bully [**?**] pertence a uma classe de algoritmos de eleição que inclui também o Ring Algorithm e algoritmos baseados em consenso como Paxos [**?**] e Raft [**?**].

Sistemas de replicação geodistribuída são amplamente estudados, com implementações de produção como Google Spanner [**?**], que utiliza relógios atômicos (TrueTime) para ordenação global, e Amazon DynamoDB [**?**], que adota consistência eventual e vector clocks.

VII. Conclusões e Trabalho Futuro

Este trabalho implementou com sucesso um sistema distribuído de log com ordenação causal através do Relógio de Lamport e coordenação via Algoritmo Bully, deployado em três regiões geograficamente distantes do Google Cloud Platform. Os experimentos conduzidos a partir de Campinas demonstraram o funcionamento correto da ordenação causal, validando a implementação dos algoritmos clássicos.

A análise das latências de rede confirmou o impacto significativo da distância geográfica, com latências variando de 19ms (São Paulo, próxima a Campinas) até 652ms (Sydney), validando a importância de estratégias de edge computing para sistemas globais. O throughput máximo observado foi de 26.19 msg/s para 50 mensagens concorrentes, com degradação severa ao atingir 100 mensagens.

Como trabalho futuro, propõe-se:

1) **Implementação completa do Bully**: Adicionar detecção de falhas e mensagens ELECTION/COORDINATOR explícitas com timeout
2) **Otimização de recursos**: Utilizar VMs com mais capacidade (e2-medium ou superior) para avaliar throughput sem gargalo de CPU
3) **Multi-leader replication**: Explorar arquiteturas multi-leader com resolução de conflitos através de CRDTs
4) **Benchmarks extensivos**: Avaliar comportamento sob diferentes padrões de carga e cenários de falha
5) **Replicação assíncrona otimizada**: Implementar replicação push ao invés de polling
6) **Comparação com Vector Clocks**: Avaliar o overhead de Vector Clocks versus Lamport Clocks

Os resultados obtidos demonstram que, embora algoritmos clássicos como Lamport Clock e Bully sejam conceitualmente simples, sua implementação em ambientes geodistribuídos reais apresenta desafios significativos relacionados a latência de rede, saturação de recursos e trade-offs entre consistência e performance. A experiência de deployar em três continentes diferentes proporcionou insights valiosos sobre as limitações físicas e práticas de sistemas distribuídos globais.

References

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558-565, 1978.

[2] H. Garcia-Molina, "Elections in a distributed computing system," *IEEE Transactions on Computers*, vol. C-31, no. 1, pp. 48-59, 1982.

[3] M. Kleppmann, *Designing Data-Intensive Applications*, O'Reilly Media, 2017.

[4] E. A. Brewer, "Towards robust distributed systems," in *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2000.

[5] C. Fidge, "Timestamps in message-passing systems that preserve the partial ordering," in *Proceedings of the 11th Australian Computer Science Conference*, 1988, pp. 56-66.

[6] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133-169, 1998.

[7] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of USENIX Annual Technical Conference*, 2014, pp. 305-319.

[8] J. C. Corbett et al., "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems*, vol. 31, no. 3, 2013.

[9] G. DeCandia et al., "Dynamo: Amazon's highly available key-value store," in *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, 2007, pp. 205-220.

[10] C. A. Astudillo, "Introdução aos Sistemas Distribuídos - Objetivos de Projeto", slides de aula MC714A, Unicamp, 2025.