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

***Resumo*—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 fundamental em sistemas distribuídos sem relógio global sincronizado. Lamport [1] introduziu relógios lógicos para estabelecer ordenação parcial consistente com causalidade. O Algoritmo Bully [2] oferece eleição de líder determinística onde o processo com maior ID assume liderança.

Este trabalho implementa ambos algoritmos em um sistema de log distribuído com três nodos, deployado em três regiões do Google Cloud Platform (Iowa, São Paulo, Sydney) para simular latências WAN realistas. Experimentos conduzidos de Campinas revelam latências de 19ms (São Paulo), 294ms (Iowa) e 652ms (Sydney).

II. Fundamentação Teórica

A. Relógio Lógico de Lamport

O Relógio de Lamport [1] estabelece ordenação parcial de eventos através de timestamps lógicos respeitando causalidade ($\rightarrow$). Cada processo P_i mantém contador C_i seguindo: (1) incremento antes de evento local $C_i := C_i + 1$; (2) envio de mensagem m com $ts(m) = C_i$; (3) ao receber m, atualização $C_j := \max(C_j, ts(m)) + 1$. Propriedade fundamental: se $e \rightarrow e'$, então $C(e) < C(e')$.

B. Algoritmo Bully e Replicação Single-Leader

O Bully [2] garante que o processo com maior ID vence eleições através de mensagens ELECTION/COORDINATOR. A arquitetura single-leader [3] direciona escritas ao líder, que replica para followers, simplificando consistência mas criando ponto único de falha.

III. Metodologia de Implementação

A. Arquitetura do Sistema

Sistema implementado em Python 3.9 com FastAPI expondo APIs REST. Arquitetura com três componentes: **LamportClock** (thread-safe com `threading.Lock`), **Server** (estado do nodo: ID, mensagens, peers) e **FastAPI Application** (endpoints para mensagens, estado e eleição). Cada mensagem contém: `id`, `content`, `lamport_timestamp`, `node_id` e `physical_timestamp`.

O relógio de Lamport usa `threading.Lock` para thread-safety, incrementando antes de eventos locais e atualizando para $\max(local, remote) + 1$ ao receber mensagens. O Bully foi simplificado: nodo com maior ID (8003 - Sydney) assume liderança automaticamente.

B. Deployment Geográfico

Deployment em GCP em três regiões: Node 1 (8001, Iowa), Node 2 (8002, São Paulo) e Node 3 (8003, Sydney). VMs e2-micro (1GB RAM) executam containers Docker com FastAPI. Startup scripts instalam Docker, obtêm IPs e configuram peers. Separação total 36.000 km: Iowa-São Paulo (8.000 km), Iowa-Sydney (13.000 km), São Paulo-Sydney (15.000 km).

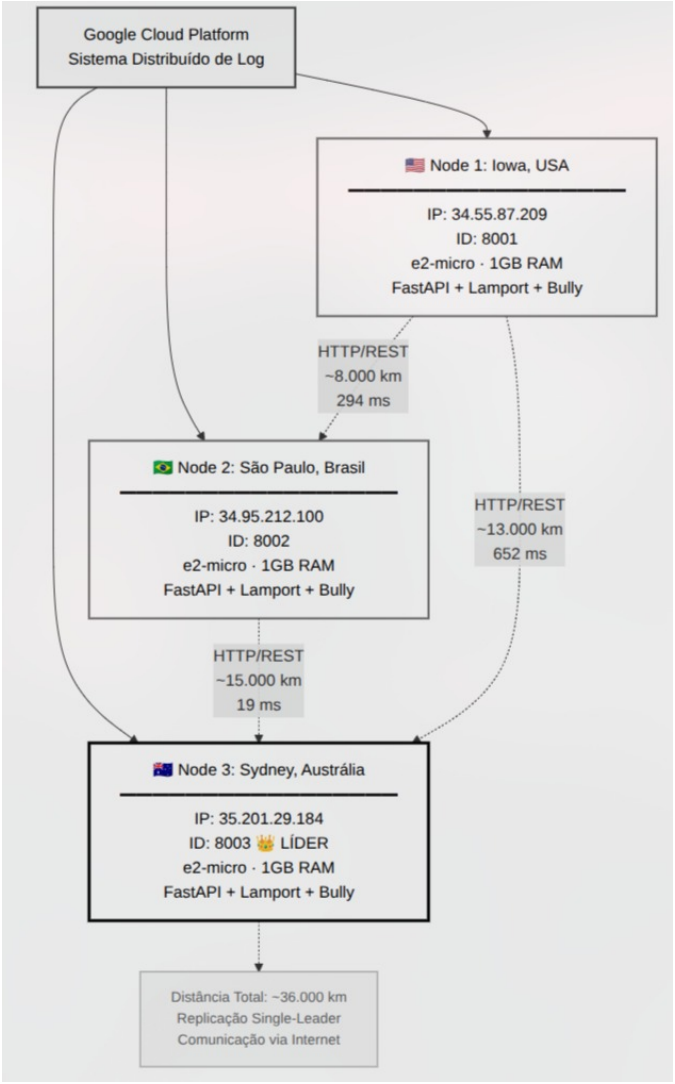

Figura 1. Arquitetura do sistema distribuído em três regiões do GCP (36.000 km).

IV. Experimentos e Métricas

Experimentos conduzidos de Campinas com scripts automatizados. Proximidade a São Paulo (90 km) resulta em latências menores.

A. Latência de Rede entre Regiões

Tabela I mostra latências HTTP medidas (5 medições por região).

Tabela I
Latências HTTP Medidas (Cliente em Campinas)

Região	Distância	Latência Média	Desvio
São Paulo	∼90 km	19 ms	±0.4 ms
Iowa	∼8.000 km	295 ms	±7.0 ms
Sydney	∼18.000 km	652 ms	±51 ms

Latência para São Paulo é 15x menor que Iowa e 34x menor que Sydney. Variação para Sydney (±51ms) sugere variações de roteamento transoceânico.

B. Throughput sob Diferentes Cargas

Throughput avaliado enviando mensagens concorrentes ao líder (Sydney). Tabela II apresenta resultados.

Tabela 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

Throughput cresce linearmente até 50 mensagens (26.19 msg/s), mas degrada 90% em 100 mensagens (2.70 msg/s). Saturação causada por: limitação de conexões HTTP concorrentes, timeout pela alta latência (600ms), e gargalo de CPU (1 vCPU). Latência média aumenta de 38.2ms para 369.8ms, indicando fila no servidor.

C. Convergência dos Relógios de Lamport

Três rodadas de escritas simultâneas (9 mensagens). Tabela III mostra estado final dos relógios.

Tabela 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 ($T_1 = 14$, $T_2 = 4$, $T_3 = 104$) reflete que o líder processa a maioria das mensagens. A Figura 2 mostra que o throughput cresce linearmente até 50 mensagens (26.19 msg/s), mas degrada 90% ao atingir 100 mensagens (2.70 msg/s). A Figura 3 evidencia crescimento exponencial da latência: de 38.2ms (carga 50) para 369.8ms (carga 100), confirmando saturação do sistema.

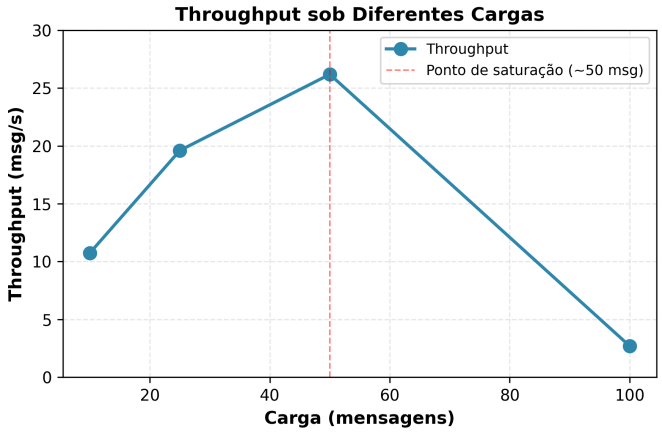

Figura 2. Throughput sob diferentes cargas, evidenciando degradação severa acima de 50 mensagens concorrentes.

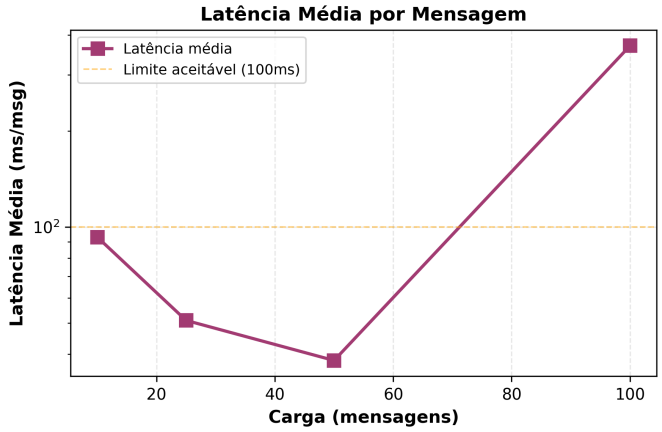

Figura 3. Latência média por mensagem (escala logarítmica), mostrando crescimento exponencial após 50 mensagens.

V. Análise dos Resultados

A. Impacto da Distância Geográfica

Distância geográfica tem impacto direto na latência. Relação quase linear observada (19ms/90km, 295ms/8.000km, 652ms/18.000km) alinha-se com velocidade de luz em fibra (∼200.000 km/s) mais latências de roteamento. Latências impõem limites fundamentais no throughput global. São Paulo próxima a Campinas demonstra benefício de edge computing.

B. Comportamento sob Carga e Consistência

Sistema estável até 50 mensagens (26.19 msg/s). Degradação severa em 100 mensagens (2.70 msg/s, 90% redução) sugere saturação por: timeout FastAPI, gargalo de CPU (1 vCPU) e latência de replicação. Valores díspares ($T_1 = 14$, $T_2 = 4$, $T_3 = 104$) confirmam que líder processa maioria das mensagens, concentrando carga na arquitetura single-leader.

Ordenação causal mantida: eventos locais incrementam relógio monotonicamente, mensagens recebidas têm timestamps maiores, relação aconteceu-antes ($\rightarrow$) preservada. Trade-off CAP [4]: implementação escolhe Consistência (Lamport) e Disponibilidade (réplicas), sacrificando Tolerância a Partições (líder único é ponto de falha).

VI. Conclusões e Trabalho Futuro

Sistema distribuído de log com ordenação causal (Relógio de Lamport) e coordenação (Algoritmo Bully) deployado em três regiões GCP validou algoritmos clássicos. Latências variaram de 19ms (São Paulo) a 652ms (Sydney), confirmando impacto da distância e benefício de edge computing. Throughput máximo de 26.19 msg/s para 50 mensagens, com degradação severa em 100 mensagens.

Trabalho futuro: (1) Bully completo com detecção de falhas; (2) VMs maiores para eliminar gargalo de CPU; (3) multi-leader com CRDTs; (4) benchmarks extensivos; (5) replicação push assíncrona; (6) comparação com Vector Clocks.

Resultados demonstram que algoritmos clássicos, embora conceitualmente simples, apresentam desafios significativos em ambientes geodistribuídos: latência de rede, saturação de recursos e trade-offs consistência-performance. Deployment em três continentes proporcionou insights sobre limitações físicas e práticas de sistemas distribuídos globais.

Referências

[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.