Balanceamento de carga com reparticionamento contínuo em sistemas com estado particionado

Douglas Pereira Luiz[1], **Odorico Machado Mendizabal**[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brazil

douglas.pereira@grad.ufsc.br, odorico.mendizabal@ufsc.br

Resumo. *Estratégias de particionamento de estado combinadas com algoritmos de rebalanceamento podem ser utilizados para equilibrar a carga em sistemas de alta vazão. É desejável manter a carga equilibrada constantemente, mas o custo da execução de um particionamento pode ser alto. Neste trabalho, apresentamos implementações de técnicas baseadas em algoritmos de particionamento de grafos para reduzir o impacto negativo causado por particionamentos e que se sobressaíram nos experimentos realizados.*

1. Introdução

Uma estratégia comum para aumentar vazão é o particionamento de estados. Entretanto, prever um bom esquema de partições pode não ser uma tarefa fácil, e estratégias que configuram o nível de paralelismo na inicialização do sistema podem não ser adequadas para cargas de trabalho que variam com o tempo [Alchieri et al. 2017]. Para maximizar os ganhos em função do paralelismo, é possível reconfigurar o particionamento durante a execução, o que pode balancear a carga de trabalho entre *threads* e reduzir o impacto causado por sincronizações devido a comandos conflitantes [Trombeta and Mendizabal 2020].

Em [Goulart et al. 2023], os autores aproveitam instantes de ociosidade do sistema para realizar o particionamento sem custo adicional. Entretanto, esse tipo de estratégia pode ser pouco adequado para sistemas de alta vazão nos quais paradas são indesejadas, preferindo-se meios de realizar rebalanceamentos de carga que não exigem interrupções prolongadas na execução de novas requisições.

Em vista disso, este trabalho apresenta técnicas para minimização do custo da reconfiguração do esquema de particionamento que não dependem da parada do sistema. As técnicas consistem em realizar o particionamento em uma linha de execução dedicada e reduzir o tempo do particionamento em si.

2. Estratégia de rebalanceamento

Este trabalho baseia-se em [Trombeta and Mendizabal 2020] e estende o protótipo desenvolvido em [Trombeta 2021], um *key-value store* que atende requisições de leitura, escrita e escaneamento. As réplicas do protótipo contêm um escalonador, *threads* trabalhadoras, e o *pattern tracker*, que é uma *thread* dedicada à construção de um grafo que representa a carga de trabalho submetida ao sistema. O particionamento do grafo, que leva a parada do escalonador, é realizado a cada Δp requisições despachadas, com Δp definido na inicialização do sistema.

A estratégia proposta não provoca longas paradas do escalonador, pois o reparticionamento é realizado continuamente, composto pelas etapas a seguir:

- Quando o *pattern tracker* observa que não há um processo de reparticionamento em andamento, ele faz uma cópia do grafo, notifica uma outra *thread*, chamada *particionador* e volta a atualizar o grafo com base nas novas requisições;
- Assim que notificado, o particionador cria um novo mapa de chaves para partições utilizando a cópia do grafo e notifica o escalonador;
- Quando o escalonador detecta a existência de um novo mapa, ele troca o mapa atual pelo mapa atualizado e indica que o rebalanceamento foi finalizado. O *pattern tracker* dará início a um novo de rebalanceamento neste momento.

O particionamento pode tomar muito tempo devido ao tamanho do grafo, que pode representar um grande número variáveis acessadas pelas requisições dos clientes. Dessa forma, o escalonador pode utilizar por muito tempo uma configuração pouco otimizada frente à carga de trabalho mais recente. Em vista disso, introduzimos uma *janela deslizante* ao *pattern tracker*, que mantém o grafo com informações somente das últimas W requisições, reduzindo o tamanho do grafo e o tempo de particionamento.

3. Avaliação Experimental

Em [Trombeta 2021], foi apresentado um protótipo de *key-value store* para avaliar a eficiência do particionamento em termos de vazão e *makespan*, doravante denominado *protótipo Base*. Estendemos essa versão, implementando as técnicas apresentadas na seção anterior[1]. A versão denominada *Non-Stop* implementa a técnica de particionamento contínuo. Essa versão não exige configurações iniciais do sistema, enquanto a versão Base exige a configuração de um intervalo de operações Δp entre rebalanceamentos. A técnica de janelas deslizantes pode ser utilizada com as duas versões, sendo adicionalmente necessário configurar o *tamanho de janela* W. O protótipo permite a configuração do número de partições/*worker threads*, além permitir a execução sem a realização de particionamentos, com o escalonamento em regime *Round Robin*, que associa novas chaves às *worker threads* de forma circular.

Para resolver o problema de corte mínimo em grafos, foi utilizado o algoritmo METIS [Karypis and Kumar 1998] nas versões que realizam a reconfiguração do particionamento. Os experimentos foram configurados com: 10^6 pares chave-valor inicialmente no sistema; 8 *worker threads*; chaves de 4 bytes; valores de 4 kbytes.

Para geração de carga, foi utilizado o *Yahoo! Cloud Serving Bechmark (YCSB)* com as cargas de trabalho D e E [Cooper et al. 2010]. A carga D foi configurada com 5×10^7 de requisições, sendo 5% escritas de novos pares (chave, valor) e 95% leituras. A carga E foi configurada com 5×10^6 de requisições, sendo 95% escaneamentos, com variação uniforme no número de chaves acessadas, entre 2 e 8, e 5% são atualizações de chaves já presentes no sistema. Os testes foram realizados em um computador com dois processadores Intel Xeon E5-2630, cada um com 2.4 GHz, 8 núcleos e 20 MB cache, e 64 GB de memória RAM DDR4. O sistema operacional usado foi o Ubuntu v22.04 de 64 bits. O programa de testes foi desenvolvido em C++17 e compilado com o *gcc v9.4.0*.

[1] `https://github.com/douglaspereira04/kvpaxos/tree/2afa8abd04abf150e4c0a171eb3df0789c2ffb88`

Foram realizados experimentos com a execução das cargas D e E e produzidos gráficos que apresentam o tempo de execução em segundos, no eixo x, e a vazão, em mil requisições/s, no eixo y. Os gráficos das Figuras 1 e 2 apresentam resultados das versões Base, *Non-Stop* (NS), *Round Robin* (RR), e execução com somente uma *worker thread* (SW). Quando configurados, os valores de Δp e W estão definidos na legenda.

Na Figura 1 são apresentados os resultados sobre a carga de trabalho D. Essa carga é composta somente por requisições que acessam uma variável, não havendo necessidade do escalonador gerenciar sincronizações entre as *threads* trabalhadoras. A implementação *SW*, com uma *thread* trabalhadora, finalizou a execução em $544s$, enquanto a versão RR finalizou em $82s$, oferecendo um bom balanceamento para essa carga de trabalho.

As versões com reconfiguração do particionamento não obtiveram ganhos de vazão suficientes para compensar o custo das operações adicionais de reparticionamento e sincronização. A implementação Base sofreu com significativas perdas de desempenho nos momentos de particionamento, com durações de até $4s$ com o grafo composto por aproximadamente 3.500.000, resultando em um tempo de execução de $85s$. A versão NS demorou ainda mais, finalizando em $\approx 89s$. As versões que utilizam janelas deslizantes tiveram reduzidas quedas de desempenho, mas os resultados de vazão não atingiram valores acima de 600.000 requisições/s, diferente das versões sem essa modificação.

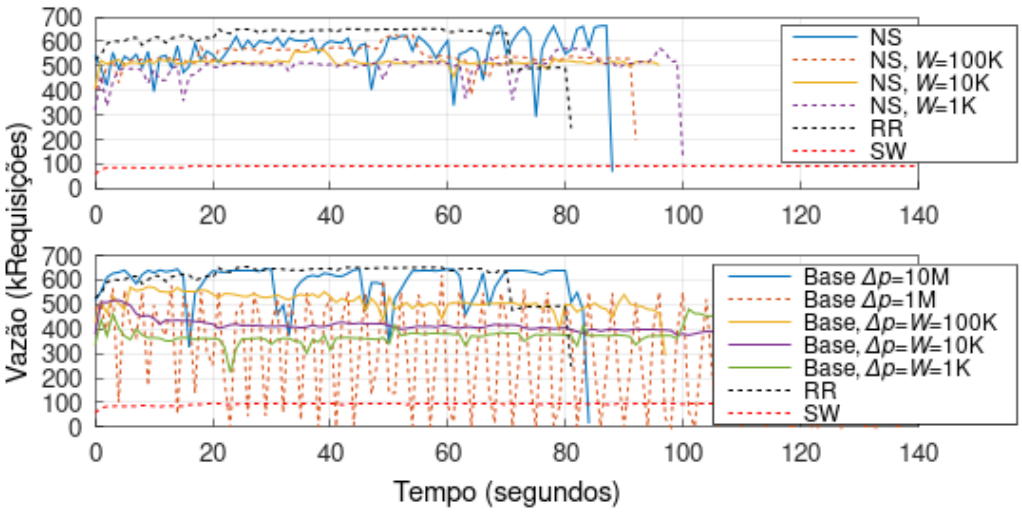

Figura 1. Resultados dos experimentos com a carga D.

A Figura 2 mostra os resultados obtidos em experimentos com a carga E, uma carga composta por um grande número de requisições multivariável que impõem sincronizações entre as *threads* trabalhadoras. O particionamento dura muito tempo devido ao tamanho do grafo, com isso, a versão NS finalizou o escalonamento sem modificar o mapeamento, terminando a execução em $\approx 980s$, tempo similar ao da versão RR.

Na versão NS, janelas de 1.000 requisições reduziram o tempo de particionamento e proporcionaram o menor tempo de execução dentre as implementações. A vazão atingiu 70.000 requisições/s e tempo de execução de $\approx 82s$. Essa configuração reduziu o tempo de execução em 46% em relação à versão Base, e em 91% em relação à versão RR. A versão Base foi favorecida por janelas de 10.000 requisições, porém a maior vazão, de aproximadamente 95.000 requisições/s, foi observada sem essa otimização.

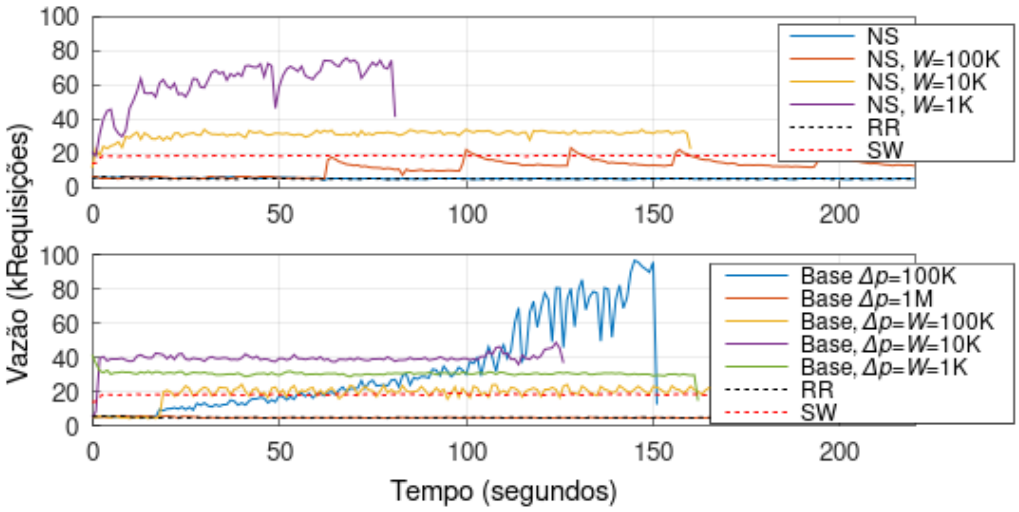

Figura 2. Resultados dos experimentos com a carga E.

4. Conclusão

Implementamos uma técnica de rebalanceamento baseada em particionamento de grafos que considera a carga de trabalho e não interrompe o escalonamento. Para otimizar o tempo de particionamento implementamos janelas deslizantes, que limitam o grafo à caracterizar somente uma porção recente do trabalho realizado.

Os experimentos sugerem um custo das técnicas quando o protótipo é submetido a cargas de trabalho dominadas por comandos sobre uma única variável e com frequente modificação no perfil de acesso às variáveis. No entanto, os ganhos foram expressivos sob cargas compostas principalmente por requisições multivariável.

Referências

Alchieri, E., Dotti, F., Mendizabal, O. M., and Pedone, F. (2017). Reconfiguring parallel state machine replication. In *Proceedings of SRDS '17*, pages 104–113.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of SoCC '10*, page 143–154.

Goulart, H., Trombeta, J., Franco, A., and Mendizabal, O. (2023). Achieving enhanced performance combining checkpointing and dynamic state partitioning. In *Proceedings of SBAC-PAD '2023*, pages 149–159.

Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, pages 359–392.

Trombeta, J. G. (2021). *Análise do uso de particionamento balanceado de grafos para explorar paralelismo em Replicação Máquina de Estados Paralela*. Monografia, Ciências da Computação, Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil.

Trombeta, J. G. and Mendizabal, O. M. (2020). Proposta para reparticionamento de estado em replicação máquina de estado paralela. In *Proceedings of COTB '20*, pages 71–73.