Avaliação de Desempenho de Bibliotecas Java para o Protocolo Paxos

Paola Pereira[1], **Rodrigo H. Müller**[1], **Odorico M. Mendizabal**[1]

[1]Centro de Ciências Computacionais – Universidade Federal do Rio Grande - FURG
Rio Grande – RS – Brazil

{paolapereira,rodrigo.muller,odoricomendizabal}@furg.br

Resumo. *Sistemas tolerantes a falhas normalmente adotam técnicas de replicação, como a Replicação Máquinas de Estados. Nesta técnica, um protocolo de ordenação garante que as réplicas executem uma mesma sequência de comandos, garantindo consistência. Devido ao bom desempenho, Paxos é um dos protocolos mais utilizados. Este artigo expõe uma análise de desempenho de bibliotecas Java para o Paxos, destacando a vazão alcançada por cada uma.*

1. Introdução

Tolerância a falhas é uma propriedade fundamental, especialmente em serviços críticos com requisitos de alta disponibilidade. Um dos meios de prover esta tolerância é através da replicação do serviço em diversas instâncias. Dentre as técnicas de replicação, a Replicação Máquina de Estados [Schneider 1990], ou replicação ativa, destaca-se pela simplicidade e garantia de consistência forte. Nesta abordagem, todos os comandos (requisições de clientes) são executados na mesma ordem por todas as réplicas, de forma determinística, garantindo que todas as réplicas passem pelos mesmos estados. Para garantir a entrega ordenada de requisições entre todas as réplicas, pode-se utilizar protocolos de difusão atômica ou de consenso. Neste sentido, o protocolo Paxos [Lamport 1998] destaca-se tanto pela sua relevância acadêmica quanto pela aplicação em ambientes corporativos [Burrows 2006, Rao et al. 2011], apresentando bom desempenho.

Embora o Paxos apresente um bom desempenho comparado com outros protocolos de ordenação, o custo de comunicação comparado com primitivas de comunicação tradicionais (não-confiáveis) é alto. Decisões de projeto para diferentes implementações do protocolo também influenciam o desempenho. Neste trabalho é feita uma análise de implementações Java de código aberto para o protocolo Paxos.

2. Paxos

Paxos [Lamport 1998] é um algoritmo de acordo que busca obter consenso de um valor entre as réplicas. Um algoritmo de consenso garante que apenas um dentre os valores propostos seja escolhido, passo fundamental para desenvolver aplicações utilizando Replicação Máquina de Estados. No Paxos existem três papéis distintos: *proposer*, *acceptor* e *learner*. O *proposer* propõe valores, que poderão ser aceitos pelos *acceptors*, desde que estes não tenham se comprometido com outro valor para a mesma instância de consenso. Após um conjunto de *acceptors* concordarem com um valor, os *learners* aprenderão o valor escolhido. Assim, um processo nunca recebe um valor, a menos que este tenha sido aprendido. Em sistemas práticos, cada réplica pode atuar em um dos papéis

ou em mais de um, simultaneamente. O protocolo de consenso atua em duas fases, geralmente chamadas de fase 1 e 2. Cada uma destas é subdividida em duas fases (1a, 1b, 2a e 2b). Na fase 1, o *proposer* envia uma requisição do tipo *accept*, informando nela um identificador único, *i*, solicitando aos *acceptors* que, naquela época, ele possa propor um valor, se já não houver algum. Quando obter o aval de um quórum de *acceptors* ($N/2+1$, sendo N o número de *acceptors*), o *proposer* passa à fase 2, na qual envia uma requisição *commit*, informando um valor. Somente após o término da segunda fase, que o valor é efetivamente replicado.

S-Paxos[1]**:** O S-Paxos [Biely et al. 2012] foi desenvolvido utilizando como base o JPaxos, uma implementação *multi-threaded* do protocolo Paxos. O S-Paxos procura aliviar a carga sobre o coordenador (*proposer* no protocolo Paxos). Para isto, as mensagens são divididas em duas categorias: disseminação das requisições dos clientes, e ordenação das mensagens. Enquanto esta última fica a cargo do coordenador, as primeiras podem ser realizadas por qualquer réplica. Desta forma, todas as réplicas recebem pedidos dos clientes, encaminhando-as às demais.

BFT-SMaRt[2]**:** O BFT-SMaRt [Bessani et al. 2014] oferece implementação do protocolo Paxos com código-fonte aberto, desenvolvido na linguagem Java. Uma de suas principais características é a opção de tolerar falhas do tipo colapso (*crash*) ou arbitrárias (bizantinas). No primeiro modo, são necessárias $2f+1$ réplicas para suportar até f falhas. Ao suportar falhas bizantinas, porém, são utilizadas $3f + 1$ réplicas. As mensagens são enviadas às réplicas através de canais seguros, utilizando um par de chaves assimétricas.

URingPaxos[3]**:** O URingPaxos [Benz et al. 2014] também implementa o protocolo Paxos em Java, com código aberto. Nesta implementação, *proposers, acceptors* e *learners* estão dispostos em uma topologia de anel. Esta abordagem favorece a vazão, dado o bom uso da largura de banda disponível. Também é possível configurar múltiplos anéis, formando grupos de *multicast*. Para efeitos de comparação, neste trabalho, é utilizado apenas um anel, caracterizando o protocolo *Ring Paxos*.

3. Avaliação

Para comparar as bibliotecas, um serviço gerenciador de travas distribuídas do tipo leitores/escritores foi implementado e replicado utilizando cada uma das implementações do Paxos (S-Paxos, BFT-SMaRt e URingPaxos). O modelo de falhas admitido é por colapso, logo, para tolerar 1 falha, todos os sistemas foram configurados com 3 réplicas (são necessários $2f + 1$ *acceptors* para tolerar f falhas).

Todas requisições do serviço possuem um comando do tipo carácter (bloquear ou liberar), um identificador da trava do tipo inteiro (admitindo-se 1 milhão de valores) e um booleano indicando se é escrita ou leitura, totalizando aproximadamente 5 bytes. Foram utilizados computadores com processador Core i5-4570@3.20 GHz e memória RAM de 8 GB, interligados por um *switch* Ethernet. Para todos os testes, a JVM (máquina virtual Java) teve o *Heap* configurado com 6GB.

O objetivo dos testes foi encontrar a vazão máxima alcançada por cada biblioteca.

[1]https://github.com/nfsantos/S-Paxos
[2]https://github.com/bft-smart/library
[3]https://github.com/sambenz/URingPaxos

Foram executados até 80 clientes por gerador de carga, começando com um computador e, gradativamente, aumentando o número até ser observada a degradação do sistema. Todos os experimentos executaram por um período de 4 minutos. Assim que o cliente obtém uma resposta, sorteia e envia outro comando para ser processado. A latência é calculada nos clientes, enquanto a vazão é medida nas réplicas. A latência, dada em segundos, corresponde à média do nonagésimo percentil de cada cliente, sendo os valores computados a cada 50 requisições. A vazão, dada em comandos por segundo, é a média entre as vazões registradas em cada réplica. A Figura 1(a) exibe a vazão máxima registrada com uso de cada uma das bibliotecas, enquanto a Figura 1(b) apresenta as latências resultantes.

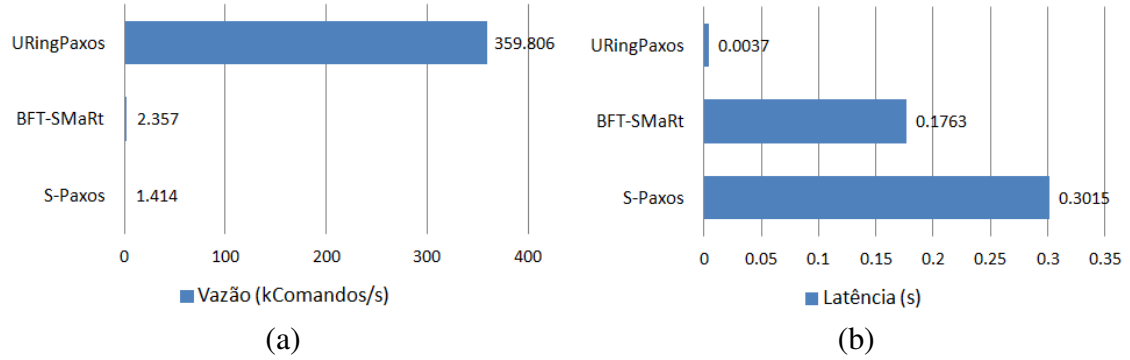

Figura 1. Comparativo de desempenho: (a) vazão máxima; (b) latência.

Com o S-Paxos observou-se uma vazão máxima de aproximadamente 1.400 comandos/s, com latência de 0,30 segundos. Após esse ponto, com a adição de mais clientes há degradação de desempenho, com o aumento da latência e sem acréscimo na vazão. O fator limitante da biblioteca parece ser a sua implementação, visto que ainda havia grande parte dos recursos disponíveis (memória, processamento e rede).

Para o BFT-SMaRt alcançou-se uma vazão máxima de 2.360 comandos/s, embora não tenha ocorrido deterioração na latência (0,18 segundos). Ao adicionar mais clientes, réplicas sofriam com escassez de memória, tornando o sistema inoperante. Conforme relatado na literatura, embora o Netty (biblioteca utilizada na comunicação do BFT-SMaRt) ofereça boa escalabilidade, ele aumenta o consumo de recursos, podendo tornar-se um gargalo de desempenho. Em [Hammerton et al. 2013], autores observaram um aumento no consumo de memória de até 3 vezes mais usando o Netty comparado com RMI. Com maior poder computacional ou ajustes na configuração, acredita-se que a vazão obtida com o BFT-SMaRt possa aumentar, visto que não houve degradação da latência nos experimentos. Em [Bessani et al. 2014], por exemplo, autores computam vazão superior a 90.000 comandos/s com o BFT-SMaRt.

Já com o URingPaxos obteve-se uma vazão máxima de 360.000 comandos/s e latência inferior a 0,004 segundos. Mesmo empregando todos os clientes disponíveis, não foi possível chegar ao ponto de degradação do sistema. À medida em que clientes eram adicionados, a vazão aumentava e a latência pouco alterava. Além disso, grande parte dos recursos permaneceram disponíveis, como memória, processamento e rede. O desempenho superior dessa implementação deve-se ao aproveitamento eficiente dos recursos. Ao contrário dos demais, o URingPaxos utiliza uma topologia em anel, gerando menos troca de mensagens, otimizando a utilização da rede e evitando custos adicionais como enfilei-

ramento em *buffers*, e emprega o Apache Thrift, diminuindo os custos com serialização e desserialização de dados. Além disso, assim como o BFT-SMaRt, suas mensagens são enviadas em lotes, diminuindo consideravelmente o número de interrupções para envio e recebimento de mensagens.

4. Conclusão

Com o trabalho realizado foi possível comparar o desempenho de três implementações Java para o protocolo Paxos: S-Paxos, BFT-SMaRt e URingPaxos. Os testes possibilitaram observar o ponto de degradação de apenas uma das bibliotecas: o S-Paxos. Em relação ao BFT-SMaRt e URingPaxos a carência de memória e número limitado de máquinas clientes, respectivamente, impediram a observação desses sistemas em estresse. Com trabalhos futuros pretende-se ampliar a investigação sobre a limitação de memória constatada no BFT-SMaRt, executar experimentos que possibilitem encontrar todos os pontos de saturação e observar o comportamento das diferentes bibliotecas em relação ao número de falhas toleradas. No presente trabalho foram realizados testes com apenas 3 réplicas (tolerando 1 falha).

Referências

Benz, S., Marandi, P. J., Pedone, F., and Garbinato, B. (2014). Building global and scalable systems with atomic multicast. In *Proceedings of the 15th International Middleware Conference*, pages 169–180. ACM.

Bessani, A., Sousa, J. a., and Alchieri, E. E. P. (2014). State machine replication for the masses with BFT-SMaRt. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '14, pages 355–362, Washington, DC, USA. IEEE Computer Society.

Biely, M., Milosevic, Z., Santos, N., and Schiper, A. (2012). S-paxos: Offloading the leader for high throughput state machine replication. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 111–120. IEEE.

Burrows, M. (2006). The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350, Berkeley, CA, USA. USENIX Association.

Hammerton, M., Trevathan, J., Myers, T., and Read, W. (2013). An evaluation of software connection methods for heterogeneous sensor networks. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 7(4):550–557.

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

Rao, J., Shekita, E. J., and Tata, S. (2011). Using paxos to build a scalable, consistent, and highly available datastore. *Proc. VLDB Endow.*, 4(4):243–254.

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