Tornando Paxos Mais Escalável com Réplicas Leitoras

Anderson P. de Paula[1] **, Gustavo M. D. Vieira**[1]

[1]DComp – CCTS – UFSCar
Sorocaba, São Paulo, Brasil

ander.pp@gmail.com, gdvieira@ufscar.br

Abstract. *Paxos is an active replication algorithm that keeps the same shared state consistently among servers that handle requests from a distributed application. In this paper we propose a reconfiguration mechanism for the Paxos algorithm that allows the incorporation of new replicas in the system without increasing significantly the cost to keep the whole system consistent. Our approach uses reader replicas, capable to answer all application requests without taking an active part in the costly operations of the Paxos algorithm. We were able to add new servers to a computation in such a way that a high load could be serviced, without compromising the application throughput.*

Resumo. *Paxos é um algoritmo de replicação ativa que consegue manter um mesmo estado compartilhado de forma consistente entre servidores que atendem a requisições de uma mesma aplicação. Neste artigo propomos um mecanismo de reconfiguração do algoritmo Paxos que permite a adição de novas réplicas sem aumentar de forma significativa o custo de manutenção da consistência do sistema como um todo. Nossa estratégia usa réplicas leitoras, que são capazes de atender a todas as requisições de aplicação sem no entanto participarem ativamente das operações custosas do algoritmo Paxos. Fomos capazes de adicionar novos servidores a uma computação de forma a absorver uma alta carga sem diminuir a vazão da aplicação.*

1. Introdução

Replicação de dados é uma estratégia amplamente empregada em sistemas distribuídos para aumentar a capacidade de processamento e prover tolerância a falhas. A *replicação ativa* [Schneider 1990] é uma estratégia de replicação voltada para manutenção de um mesmo estado compartilhado entre servidores que atendem requisições de uma mesma aplicação, sendo esses servidores chamados de *réplicas*. Dentre os vários algoritmos de replicação, um dos mais amplamente usados e estudados atualmente é o algoritmo Paxos [Lamport 2006]. De forma geral, é incomum encontrar aplicações onde a parte principal do processamento acontece através de replicação ativa devido ao fato que essa estratégia possui um custo considerável em termos do número de mensagens trocadas, o que limita a escalabilidade do sistema além de algumas poucas réplicas.

Treplica [Vieira and Buzato 2008, 2010] é uma biblioteca de replicação projetada para prover uma forma simples e orientada a objetos de se construir aplicações altamente confiáveis. Treplica permite que o projetista de aplicação pense em termos de operações com semântica sequencial, similar àquela encontrada em sistemas de processamento transacional. Utilizando essa biblioteca a aplicação resultante preserva a consistência de uma aplicação centralizada e adiciona a tolerância à falhas de uma aplicação distribuída.

Apesar da maior confiabilidade, construir uma aplicação somente com replicação ativa potencialmente limita o quanto que essa aplicação pode tirar proveito dos ganhos de escala advindos de ser uma aplicação distribuída. Resultados experimentais mostram o impacto negativo do aumento da escala no desempenho da implementação de Paxos encontrada em Treplica [Vieira and Buzato 2009]. Gostaríamos de ser capazes de não só tornar a capacidade de processamento proporcional ao número de servidores empregados, mas também de variar essa capacidade dinamicamente em reposta às mudanças da demanda gerada. Dessa forma, teríamos aplicações com a simplicidade de programação de aplicações centralizadas e características de desempenho de aplicações distribuídas.

Neste trabalho propomos um mecanismo de reconfiguração do algoritmo Paxos que permite a adição de novas réplicas sem aumentar de forma significativa o custo de manutenção da consistência do sistema como um todo. A nossa proposta tem como base fundamental o estabelecimento de *réplicas leitoras*, que são capazes de atender a todas as requisições de aplicação sem, no entanto, requerer acesso à memória persistente e sem participarem ativamente das operações custosas do algoritmo Paxos. Experimentalmente, fomos capazes de adicionar novos servidores a uma computação de forma a absorver uma alta carga sem diminuir a vazão da aplicação. As principais contribuições deste artigo são a proposta de utilização de réplicas leitoras e um mecanismo para o provisionamento eficiente dessas réplicas.

2. Paxos com Réplicas Leitoras

A ideia principal da abordagem proposta é utilizar réplicas que não participem do processo de decisão de instâncias de consenso. Isso é feito com a adoção de *réplicas leitoras*, que são réplicas onde apenas parte dos agentes do algoritmo Paxos estão executando. Para maior clareza de exposição, quando necessário, chamaremos as réplicas contendo todos os agentes ativos de *réplicas votantes*. Para suportar a adaptação elástica a novos perfis de desempenho, desenvolvemos um mecanismo para o *provisionamento de réplicas*.

2.1. Réplicas Leitoras

Réplicas leitoras são réplicas onde apenas os agentes proponente (*proposer*) e aprendiz (*learner*) [Lamport 2006] do Paxos estão executando. Desta forma, do ponto de vista do conjunto de processos que implementam o algoritmo, uma réplica leitora é capaz apenas de propor operações a serem aplicadas no estado replicado e de aprender operações decididas pelo conjunto de receptores. Do ponto de vista do cliente da aplicação replicada um réplica leitora se comporta como uma réplica votante: ela atende requisições de qualquer tipo e garante a execução atômica das mesmas.

As réplicas leitoras não assumem um papel fundamental na execução do algoritmo Paxos, porém elas se integram de forma consistente com a operação das réplicas votantes por meio de suas funções fundamentais: propor e aprender requisições de escrita. As réplicas leitoras propõem novas requisições a serem executadas em nome de seus clientes através de seu agente proponente. O proponente encaminha a operação ao coordenador (*coordinator*) que por sua vez decide, em conjunto com os receptores (*acceptors*), a ordem da mesma através de uma rodada de Paxos. Uma vez que a decisão é alcançada, a mesma é difundida para o resto do sistema. Nesse momento o agente aprendiz da réplica leitora toma conhecimento da decisão e atualiza o seu estado interno, sem a participação ativa do coordenador ou de qualquer receptor.

Uma consequência importante do uso de réplicas leitoras é que essas réplicas, consistentemente com as funções que elas assumem no algoritmo Paxos, não precisam de memória persistente para sua operação [Lamport 2006]. Na nossa proposta de réplicas leitoras decidimos não fazer o registro estável de forma a remover completamente a escrita em disco do caminho crítico de execução. É interessante observar que a escrita eliminada ocorre somente quando a réplica leitora atualiza o seu estado de acordo com as propostas decididas pelos receptores das réplicas votantes. Desta forma, as réplicas leitoras conseguem manter seu estado atualizado com as réplicas votantes com um custo mínimo. Elas também são capazes de processar requisições de escrita com um custo similar àquele gerado pelas réplicas votantes ao executar as mesmas requisições. Podemos argumentar que esse custo é menor, na medida que as réplicas leitoras aliviam as réplicas votantes do custo de manter as conexões abertas com os clientes. É concebível ainda uma configuração onde as réplicas votantes não entrem em contato com os clientes, sendo essa operação completamente delegada às replicas leitoras.

As réplicas leitoras funcionam então como uma espécie de cache *write-through* distribuído. O estado replicado na memória destas réplicas permite atender diretamente as requisições de leitura dos clientes, enquanto as requisições de escrita são repassadas ao receptores. Podemos ver claramente que a taxa de acerto desse cache está diretamente ligada à proporção de operações de leitura geradas pelos clientes e que a vazão de operações de leitura tem o potencial de crescer linearmente com o número de réplicas leitoras disponíveis.

2.2. Provisionamento de Réplicas Leitoras

É possível utilizar os mecanismos tradicionais de Treplica para provisionar uma nova réplica leitora. Em resumo, uma réplica que se integra ao sistema pela primeira vez ou após uma falha demorada deve recuperar o seu estado. Esse processo acontece através de um mecanismo de preenchimento de lacunas, que observa que não pode executar novas requisições de escrita sem antes executar as requisições anteriores [Vieira and Buzato 2010]. Esse procedimento é voltado para reparar pequenas interrupções e não a recuperação do estado completo de uma réplica. Em particular, no caso de uma réplica leitora sem estado persistente, o tamanho dessa recuperação pode ser muito grande em termos do número de *requisições* a serem reexecutadas, pois ela sempre parte do estado inicial vazio.

Foi necessário então criar um procedimento de provisionamento de réplicas, de forma a permitir o rápido início de uma réplica leitora. Esse mecanismo não é necessariamente exclusivo de réplicas leitoras e pode ser aplicado a réplicas votantes. Porém, neste primeiro momento, ele tira proveito do fato dessas réplicas não terem memória persistente. Em particular, a adição ou remoção de uma réplica leitora não altera o número de receptores executando o algoritmo, não havendo necessidade de se realizar um reconfiguração custosa [Lamport et al. 2010].

2.2.1. Protocolo

Para que a agregação de uma nova réplica seja produtiva, é preciso realizar a transferência completa de um estado inicial que deve ser clonado de outra réplica já ativa, votante ou leitora. Por questões de garantia de consistência, a transferência de estado é uma tarefa

síncrona. Essa é uma atividade potencialmente custosa para o desempenho de Paxos, pois estamos bloqueando, temporariamente, a participação de uma réplica no processo de decisão de instâncias de consenso. Visando minimizar a degradação de desempenho causada pela necessidade de transferência de estado, criamos um protocolo simples e eficaz para reger a junção de novas réplicas leitoras à computação, conforme ilustrado na Figura 1. Durante o processo de transferência de estado, supomos que a réplica interessada em receber um novo estado é sempre uma réplica leitora e que a réplica que fornece esse estado pode ser tanto uma réplica leitora quanto votante. No restante da seção chamaremos essas réplicas de réplicas doadoras.

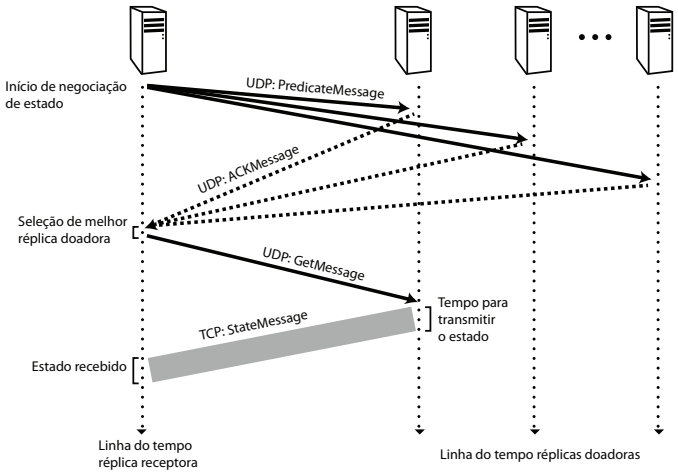

Figura 1. Protocolo de Transferência de Estado

Supomos que é responsabilidade da réplica leitora encontrar uma réplica disposta a transferir seu estado. A réplica leitora difunde uma mensagem (`PredicateMessage`) com o valor da última instância de consenso decidida. As réplicas que recebem a mensagem solicitando o estado enviam uma mensagem (`ACKMessage`) somente para o remetente do pedido de estado, com as informações sobre o seu estado atual (última instância de consenso e um indicador do tipo de réplica: leitora ou votante).

Dado que o estado de uma aplicação pode ser tão grande quanto a capacidade de memória de uma réplica e que todas as mensagens de Treplica são trocadas via protocolo UDP, optamos por aumentar a vazão da transferência de estado usando o protocolo TCP. Somente o estado é enviado via TCP, todas as outras mensagens utilizam a comunicação UDP nativa de Treplica. Sendo assim, a réplica leitora abre um *socket* TCP e envia uma mensagem (`GETMessage`) para a réplica doadora com o endereço do *socket* TCP para envio de estado. A réplica doadora bloqueia suas atividades, estabelece a conexão TCP com a réplica leitora e transfere seu estado.

3. Desempenho Preliminar

Nesta seção apresentamos resultados preliminares de desempenho de um sistema composto por uma combinação de réplicas votantes e leitoras em comparação a um sistema composto por apenas réplicas votantes. O resultado do experimento pode ser visto na Figura 2. Em relação à vazão do sistema, podemos observar que a adição de réplicas leitoras permitiu, logo a partir da primeira réplica, que o sistema passasse a atender as 8000

requisições/segundo geradas. A configuração composta totalmente por réplicas votantes conseguiu atingir apenas um valor em torno de 7700 requisições/segundo.

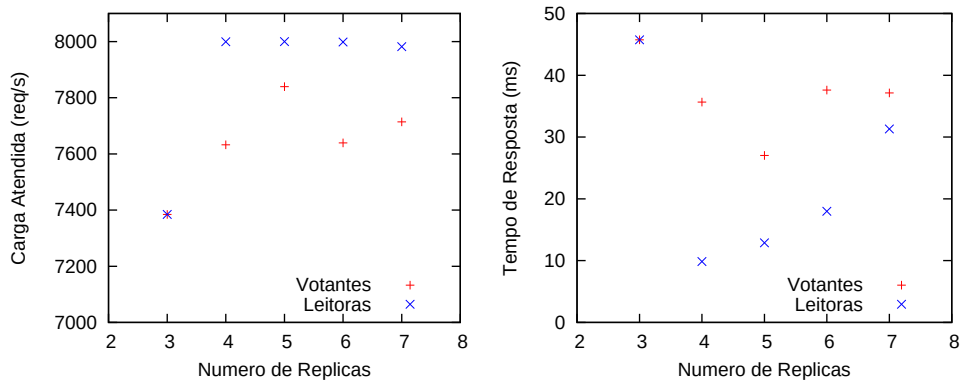

Figura 2. Aumento de Escala (8000 op/s)

4. Conclusão

Neste trabalho nós mostramos um mecanismo simples e eficaz para a escalabilidade de aplicações que usam replicação ativa, em particular o algoritmo Paxos. Acreditamos que uma maior escalabilidade pode ampliar as oportunidades de aplicação de replicação ativa para novos contextos. Especificamente, aplicações Web com uma parcela considerável de requisições de leitura podem se beneficiar da solução proposta. A aplicação nesses contextos depende ainda da adoção de uma política de reconfiguração eficiente, que esperamos desenvolver no prosseguimento desta pesquisa.

Agradecimentos

Os autores gostariam de agradecer ao Laboratório de Sistemas Distribuídos do Instituto de Computação da UNICAMP pela cessão das máquinas utilizadas nos experimentos aqui apresentados.

Referências

Lamport, L. (2006). Fast Paxos. *Distrib. Comput.*, 19(2):79–103.

Lamport, L., Malkhi, D., and Zhou, L. (2010). Reconfiguring a state machine. *SIGACT News*, 41(1):63–73.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319.

Vieira, G. M. D. and Buzato, L. E. (2008). Treplica: Ubiquitous replication. In *SBRC '08: Proc. of the 26th Brazilian Symposium on Computer Networks and Distributed Systems*, Rio de Janeiro, Brasil.

Vieira, G. M. D. and Buzato, L. E. (2009). The performance of Paxos and Fast Paxos. In *SBRC '09: Proc. of the 27th Brazilian Symposium on Computer Networks and Distributed Systems*, pages 291–304, Recife, Brasil.

Vieira, G. M. D. and Buzato, L. E. (2010). Implementation of an object-oriented specification for active replication using consensus. Technical Report IC-10-26, Institute of Computing, University of Campinas.