Validação de Consistência em Aplicações Replicadas

Vinícius Lopes da Silva
DComp — CCTS — UFSCar
Sorocaba, São Paulo
vinicius.lopes.silva89@gmail.com

Gustavo Maciel Dias Vieira
DComp — CCTS — UFSCar
Sorocaba, São Paulo
gdvieira@ufscar.br

RESUMO

Esta pesquisa de iniciação científica visou melhorar a confiabilidade da biblioteca de replicação Treplica, no contexto do desenvolvimento de aplicações web. Com esse objetivo foi criada uma ferramenta de validação de consistência de dados e a mesma foi usada para fazer uma avaliação comparativa da confiabilidade de Treplica. A base desta comparação foi a substituição do algoritmo de consenso Paxos usado por Treplica pelo mecanismo de ordenação baseado em sincronia virtual JGroups. Como resultado foi possível averiguar que a confiabilidade da implementação atual de Treplica/Paxos é superior ao JGroups.

Palavras-chave

Middleware, Replicação, Paxos, Tolerância a falhas, Ordenação Total, Sistemas distribuídos

1. INTRODUÇÃO

A web tornou-se ferramenta essencial para as empresas que desejam manter-se no mercado. Muitas delas disponibilizam serviços que servem para entreter e facilitar o dia a dia dos usuários, como por exemplo transferências de dinheiro entre contas bancárias, Massively Multiplayer Online Games (MMOG), armazenamento de e-mail, entre outros. Por serem aplicações interativas, esses sistemas exigem tempo de resposta reduzido, servem um grande número de clientes e o seu mal funcionamento pode parar completamente processos de negócios vitais. Mesmo assim, falhas são frequentes. Por várias razões, sistemas web são implementados como sistemas distribuídos. Umas das razões é dividir a carga gerada por um parque muito grande de usuários entre servidores especializados em atividades distintas. Daí vem a conhecida arquitetura em três camadas de aplicações web, com um servidor HTTP frontal que repassa requisições a um servidor de aplicações que faz uso de um sistema gerenciador de banco de dados. Desta forma, no núcleo destas aplicações reside um componente de banco de dados, responsável por armazenar o seu estado. Normalmente, esse componente é implementado por um sistema gerenciador de banco de dados (SGBD) centralizado, tornando a armazenagem persistente de dados um gargalo de desempenho e um ponto único de falhas. A replicação de dados pode ser usada para resolver os problemas causados por esta centralização, mas esta é uma solução raramente usada, exceto em aplicações muito específicas. Por trás da baixa adoção da replicação reside o fato de que é usualmente muito difícil replicar dados de forma consistente e ao mesmo tempo preservar o desempenho de um sistema centralizado [1].

As mesmas mudanças que nos tornaram tão dependentes de aplicações web também trouxeram novas oportunidades a serem exploradas. *Hardware* de prateleira começou a ser organizado em poderosos aglomerados de estações de trabalho [2]. Esses aglomerados são construídos com partes facilmente obtidas e quando software livre, revelam-se um ambiente de custo atraente para a execução de aplicações paralelas e replicadas de alto desempenho. O desafio que se apresenta agora é combinar aglomerados computacionais e replicação de dados para aumentar a disponibilidade de aplicações web, igualando e melhorando o desempenho de sistemas centralizados e não confiáveis.

A biblioteca de replicação Treplica [3, 4] é uma abordagem para resolver esse problema. Treplica simplifica o desenvolvimento de aplicações altamente disponíveis ao tornar transparente a complexidade de se lidar com replicação e persistência de dados. Esta complexidade não é desprezível [5], e Treplica define uma forma atraente de fatorar esses requisitos em uma interface de programação simples de entender. Treplica se situa a meio caminho entre a flexibilidade de baixo nível de um sistema de comunicação em grupo [6] baseado em sincronia virtual [7, 8] e os vastos recursos de processamento de dados de um SGBD. A principal característica de Treplica é a ideia de apresentar ao programador web uma abstração de programação unificada para replicação e persistência, propondo o uso de consenso [9] como a fundação para a construção desta ferramenta unificada.

Esta pesquisa de iniciação científica visou melhorar a confiabilidade da biblioteca de replicação Treplica, no contexto do desenvolvimento de aplicações web. Com esse objetivo foi criada uma ferramenta de validação de consistência de dados e a mesma foi usada para fazer uma avaliação comparativa da confiabilidade de Treplica. A base desta comparação foi a substituição do algoritmo de consenso Paxos [11, 12] usado por Treplica pelo mecanismo de ordenação

baseado em sincronia virtual JGroups [14]. Como resultado foi possível averiguar que a confiabilidade da implementação atual de Treplica/Paxos é superior ao JGroups. Mesmo apresentando confiabilidade superior, a estratégia de avaliação adotada permitiu identificar e corrigir erros latentes na implementação atual de Treplica/Paxos.

2. TREPLICA

Treplica foi projetada para prover uma forma simples e orientada a objetos de se construir aplicações altamente confiáveis. Estas aplicações podem se estender ao sistema inteiro ou se restringir à subsistemas onde o desempenho, consistência e confiabilidade são cruciais. Para alcançar esse objetivo, Treplica decompõe o problema de se implementar replicação em componentes com interfaces simples e bem definidas. Desta forma, um desenvolvedor que deseja implementar uma aplicação distribuída não precisa pensar em termos de processos, mensagens e falhas. Ao invés, ele pensa sobre a execução das operações da aplicação, transições de uma máquina de estados replicada, que são disparadas por eventos disponibilizados através de uma fila persistente assíncrona [4].

A principal decisão de projeto por trás do Treplica é que o programador pode considerar a sua aplicação como não tendo estado, ficando a durabilidade da mesma garantida pela biblioteca. Esta decisão é suportada pela observação que os mesmos requisitos de replicação ativa podem ser usados para prover um mecanismo simples e poderoso de persistência. A replicação ativa exige que a aplicação execute ações que modificam o seu estado de forma determinista. Estas ações são então propagadas, na mesma ordem, para todas as réplicas de um serviço que as reexecutam localmente. Dentro desta organização, as ações não são apenas enviadas para as outras réplicas, mas arquivadas em memória estável [10]. Desta forma, é possível recuperar de uma falha reexecutando o histórico de ações. O determinismo garante que após cada recuperação a aplicação reiniciará com o mesmo estado que possuía antes da falha.

Para suportar a replicação ativa, Treplica se concentra em algoritmos de ordenação total baseado em consenso [9] para o modelo de falhas falha-e-recuperação. O algoritmo Paxos [11] e suas variantes [12] são especialmente aptos para esse uso, pois foram criados especificamente para suportar replicação ativa. No entanto, qualquer mecanismo de troca de mensagens totalmente ordenadas pode ser usado como base para a construção de suas abstrações básicas.

3. JGROUPS

A difusão totalmente ordenada usada por Treplica é comumente associada a ferramentas de comunicação em grupos. A ferramenta Isis [6] introduziu muitas ideias que influenciaram outras ferramentas, incluindo a noção de sincronia virtual [7,8]. Implementações mais recentes destas ideias podem ser encontrados em Ensemble [13], JGroups [14], Spread [15] e Appia [16]. Neste projeto, a ferramenta escolhida para ser integrada ao Treplica e substituir o algoritmo Paxos foi o JGroups. Apesar de não ter sido a única possibilidade de ferramenta para substituir o Paxos, o JGroups foi escolhido devido a sua adoção em grandes projetos como o JBoss e também por apresentar uma extensa documentação, além de um grupo de discussões ativo.

O JGroups é um *middleware* que garante a comunicação em grupo de maneira confiável [14]. Os membros (nós) são processos que fazem parte de um grupo (*cluster*). O JGroups utiliza um paradigma de comunicação orientado a troca de mensagens, permitindo que um membro possa enviar mensagens para todos os outros do mesmo grupo ou enviar mensagens diretamente para um único membro. O JGroups monitora todos os membros que participam de um grupo, notifica todos os membros do grupo quando um novo membro entra ou sai do grupo e também quando um membro recebe uma nova mensagem. Para que uma aplicação possa enviar e receber mensagens usando o JGroups, ela precisa criar um *channel* (cujo conceito é similar ao de um socket). Esse *channel* será conectado a um grupo e possui um endereço único.

4. VALIDAÇÃO DE CONSISTÊNCIA

A validação formal da implementação de um *middleware* como o Treplica é uma tarefa difícil e que não possui ferramentas e procedimentos gerais definidos. Por isso, na iniciação científica foi adotada uma forma de validação mais simples, que consiste em criar um teste de longa duração que avalie a consistência de uma aplicação construída usando o Treplica.

Uma aplicação distribuída que use replicação é dita consistente se, após um certo tempo de execução, todas as réplicas do serviço possuem o mesmo estado. A consistência desta aplicação é resultado direto da correção do sistema de replicação subjacente. Desta forma, a criação de uma ferramenta de testes de longa duração envolve o desenvolvimento de uma aplicação onde é fácil determinar uma violação de consistência e a criação de um ambiente de execução que seja capaz de executar esta aplicação de forma controlada. A ideia adotada desde o início do projeto era a de criar uma ferramenta autônoma e simples de usar, que possa ser empregada sempre que Treplica for adaptada a um novo ambiente ou modificações sejam realizadas em sua implementação.

5. RESULTADOS

Diversos testes usando a ferramenta desenvolvida foram realizados para o JGroups e o Treplica. Nesta seção serão apresentadas a ferramenta de testes e os experimentos que foram relevantes para se concluir algo a respeito do funcionamento de ambos os sistemas através da aplicação criada. Na nomenclatura de comunicação em grupo, os membros de um grupo podem ser chamados de nós, entretanto, nesta seção eles serão chamados de réplicas, já que sua função é justamente a de manter o estado replicado.

5.1 Implementação da Ferramenta de Testes

A ferramenta desenvolvida consiste em trocas de mensagens em um ambiente de comunicação em grupo, com um limite fixo N na quantidade de membros que podem fazer parte do grupo. Portanto, o grupo não pode ter um número de membros maior do que N, entretanto, é permitido que haja um número de membros inferior a N, desde que esse número não seja inferior a $N/2 + 1$, caso seja, a aplicação interrompe o envio de mensagens e fica aguardando um número de membros que seja maior ou igual a $N/2 + 1$ para que volte a trocar mensagens entre as réplicas do grupo. É importante que a aplicação não continue executando caso haja

menos de N/2 + 1 nós. Caso essa regra seja violada, é possível que haja inconsistência sobre o estado mantido entre os diferentes nós e não haveria como contornar tal situação de maneira elegante. Garantindo que sempre haja mais da metade dos nós em execução, é possível reverter situações onde uma minoria deles armazenam um estado diferente da maioria.

As mensagens enviadas para os membros do grupo são geradas aleatoriamente e tem um tamanho fixo de três caracteres, é importante ressaltar que esse tamanho foi definido arbitrariamente e não implica em perda de desempenho ou comprometimento da consistência do estado, podendo ser alterado conforme as necessidades do desenvolvedor. As réplicas mantém um estado que contém o histórico de todas as mensagens que foram recebidas, as mensagens enviadas por uma réplica também são enviadas para a réplica que enviou a mensagem em questão, como pode ser observado na Figura 1.

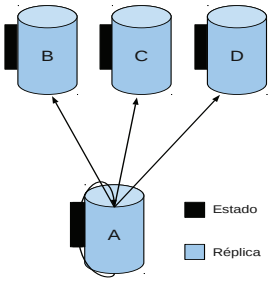

Figura 1: Esquema representando um ambiente de comunicação em grupo onde a réplica A envia mensagem para todas as outras réplicas do grupo e todas as réplicas possuem o mesmo estado

Devido ao fato desse histórico de mensagens crescer muito rapidamente, foi utilizada uma abordagem para reduzir a quantidade de memória usada pela aplicação. Para conseguir isso, foi utilizado o CRC-32 que gera um hash do histórico de mensagens com um tamanho fixo de 32 bits. Desta forma, sempre que uma mensagem é recebida, é calculado um novo hash para o histórico de mensagens recebidos, se o hash de todas as réplicas for igual, então é dito que a aplicação está consistente. Sempre que a aplicação é executada, deve-se informar a frequência com que as mensagens serão enviadas e o tempo em segundos pelo qual a aplicação executará. Ao término da execução, as réplicas informam o hash final e o número de operações que foram realizadas. O tempo de duração de um experimento pode ser grande, portanto, deseja-se saber se a consistência do estado foi comprometida antes de chegar ao fim do experimento. Por isso, durante a sua execução, as réplicas exibem o seu hash a cada 100 mensagens recebidas, permitindo saber se a consistência foi violada ou não. Caso seja necessário, é possível abortar o experimento e fazer as correções necessárias.

5.2 Validação de JGroups

Foram feitas três réplicas para a execução dos experimentos com o JGroups, sendo duas delas em uma máquina A e uma na máquina B. As três réplicas executariam por 18000s (5h), sendo que todas as réplicas possuíam um mesmo vazão de 700 msg/s. Nesse experimento o JGroups durou aproximadamente 2h, sem a introdução de falhas. Após esse tempo começaram a surgir mensagens de erro informando que não havia como retransmitir as mensagens para os outros nós devido a ausência de memória. Isso mostra que haviam limitações de *hardware*, portanto, o vazão foi reduzido e as réplicas passaram a ter um vazão de 300 msg/s. Após 1h foi verificado que as réplicas continuavam consistentes, então começou-se a introduzir falhas no sistema. Especificamente, matava-se uma das réplicas em execução de forma a manter o grupo em execução. Após um certo tempo do grupo em execução com um membro a menos, inseria-se esse novo membro no grupo. Esse então obtinha o estado atual do grupo e executava normalmente com os outros membros, mantendo assim a consistência do estado em todas as réplicas.

Entretanto, quando foi introduzida a falha em que o cabo de rede é retirado da máquina física a consistência foi violada. A réplica sai do grupo ao qual pertencia e cria um novo grupo onde participarão apenas as réplicas que estão na mesma máquina física, caso o número de membros desse novo grupo seja maior ou igual a N/2 + 1. Esse grupo então vai executar como se nada de errado tivesse ocorrido, ou seja, eles executarão como se estivessem em rede ainda. Teoricamente, quando o cabo de rede fosse colocado novamente na máquina física, as réplicas deveriam voltar a formar o mesmo grupo de maneira a tornar o estado consistente, mas isso não ocorreu. Foram pesquisadas informações no grupo de discussões do JGroups, mas a resposta obtida do autor do programa foi a de que deve ser um bug do próprio JGroups. A pilha de protocolos, montada em XML e que possui uma camada que garante a ordenação total e outra que garante a sincronia virtual, não havia sido testada apropriadamente e correspondia a uma configuração não suportada. Ou seja, para replicação usando ordenação total de mensagens o JGroups não oferece a confiabilidade mínima necessária.

5.3 Validação de Treplica/Paxos

O Treplica utiliza um paradigma de programação orientado a estados, diferente do que ocorre no JGroups, o que pode causar um pouco de confusão para desenvolvedores que não estão acostumados com esse paradigma. Os experimentos desenvolvidos para o Treplica possuíam o mesmo objetivo daqueles que foram feitos para o JGroups. O ambiente de testes foi o mesmo, mas diferente dos experimentos usando o JGroups onde o vazão era de mais de 100 msg/s, nos experimentos feitos usando o Treplica, o vazão foi reduzido para 10 msg/s nas três réplicas. Isto ocorre pois, usando Paxos, o Treplica não chega a enviar mais de 100 mensagens por segundo, independente do valor de frequência que for informado.

Os experimentos executaram normalmente, mantendo a consistência do estado entre todas as réplicas por duas horas sem a introdução de falhas e também ao introduzir a falha de matar uma das réplicas e colocá-la novamente no grupo. Ao introduzir a falha de retirar o cabo de rede, novamente foi observado um problema. Ao perder a conexão a réplica interrompia sua execução, mas ao colocar novamente o cabo de rede na máquina física a réplica não identificava que a má-

quina física havia sido reconectada. Foi identificada então uma limitação do Treplica em relação a sua confiabilidade.

Foi necessário fazer uma correção no Treplica, para resolver o problema identificado. A abordagem anterior adotada pelo Treplica era a de lançar uma exceção para qualquer erro relacionado ao envio de mensagem. A modificação feita permite tratar a exceção e, quando ocorre uma exceção relacionada a E/S ao enviar uma mensagem, o Treplica tenta reenviá-la, caso o cabo de rede seja conectado a tempo, então o Treplica envia a mensagem para os membros do grupo e a réplica que havia sido interrompida recebe o novo estado do grupo, mantendo assim a consistência do estado entre todas as réplicas. Essa abordagem é muito importante pois torna transparente para usuários e aplicações possíveis falhas externas que são comuns em um ambiente de rede.

6. CONCLUSÃO

A abstração utilizada pelo JGroups é baixa se comparada ao Treplica. No JGroups é necessário dar um nome para o grupo criado e é necessário conectar o membro a esse grupo em questão através do *channel* ou de um bloco de construção, consequentemente, os mesmos devem ser gerenciados pela própria aplicação. Ao desenvolver a aplicação para o Treplica, mudou-se o paradigma de desenvolvimento. Enquanto no JGroups utiliza-se um paradigma orientado a troca de mensagens, o Treplica utiliza um paradigma orientado ao estado, o que pode levar mais facilmente a erros conceituais que acarretam na inconsistência do estado. Por outro lado, o Treplica garante mais transparência. Além disso, para a aplicação desenvolvida, o JGroups apresentou bugs, não permitindo que ele suportasse as possíveis falhas introduzidas no sistema. Teoricamente isso deveria ser suportado modificando a pilha de protocolos, mas a pilha montada não é suportada pelo JGroups. O Treplica por outro lado consegue tratar as falhas apresentadas de maneira transparente, permitindo que o desenvolvedor não se preocupe com possíveis falhas externas, aumentando sua produtividade. A ferramenta de testes desenvolvida serviu para validar os ambientes de desenvolvimento tanto do Treplica quanto do JGroups, mas ela pode ser utilizada para validar outras ferramentas que se propõe a garantir consistência em aplicações distribuídas.

7. BIBLIOGRAFIA

[1] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data, pages 173–182, New York, NY, USA, 1996. ACM Press.

[2] T. Sterling, D. J. Becker, J. E. Dorband, D. Savarese, U. A. Ranawake,and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In Proceedings of the 24th International Conference on Parallel Processing, pages I:11–14, 1995.

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

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

[5] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: an engineering perspective. In PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, pages 398–407, New York, NY, USA, 2007. ACM Press.

[6] K. P. Birman. The process group approach to reliable distributed computing. Commun. ACM, 36(12):37–53, 1993.

[7] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. ACM Trans. Comput. Syst., 5(1):47–76, 1987.

[8] R. Friedman and R. van Renesse. Strong and weak virtual synchrony in Horus. In SRDS '96: Proceedings of the 15th Symposium on Reliable Distributed Systems (SRDS '96), page 140, Washington, DC, USA, 1996. IEEE Computer Society.

[9] M. Barborak, A. Dahbura, and M. Malek. The consensus problem in fault- tolerant computing. ACM Comput. Surv., 25:171–220, June 1993.

[10] A. D. Birrell, M. B. Jones, and E. P. Wobber. A simple and efficient implementation of a small database. In SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles, pages 149–154, New York, NY, USA, 1987. ACM Press.

[11] L. Lamport. The part-time parliament. ACM Trans. Comput. Syst., 16(2):133–169, 1998.

[12] L. Lamport. Lower bounds for asynchronous consensus. Distributed Computing, 19(2):104–125, June 2006.

[13] R. van Renesse, K. P. Birman, M. Hayden, A. Vaysburd, and D. Karr. Building adaptive systems using Ensemble. Softw. Pract. Exper., 28(9):963–979, 1998.

[14] B. Ban. Design and implementation of a reliable group communication toolkit for java. Technical report, Cornell University, 1998.

[15] Y. Amir, C. Danilov, and J. R. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In DSN '00: Proceedings of the 2000 International Conference on Dependable Systems and Networks, pages 327–336, Washington, DC, USA, 2000. IEEE Computer Society.

[16] H. Miranda, A. Pinto, and L. Rodrigues. Appia: A flexible protocol kernel supporting multiple coordinated channels. In ICDCS '01: Proceedings of the The 21st International Conference on Distributed Computing Systems, pages 707–710, Washington, DC, USA, Apr. 2001. IEEE Computer Society.