Instituto de Matemática e Estatística - USP

Trabalho final - MAC5742

Projeto RAMCloud

Autor:
Bruno Padilha

Prof.:
Alfredo Goldman vel Lejbman

26 de junho de 2015

Resumo

Desde o início dos anos 80 o disco magnético, também conhecido por disco rígido, hard disk ou HD, é o principal meio para armazenamento de dados em massa. Embora sua capacidade de armazenamento tenha aumentado exponencialmente ao longo dos anos, acompanhado a evolução dos outro componentes de um PC, seu tempo de acesso e sua taxa de transferência de dados pouco evoluíram. Com a crescente demanda por serviços que acessam grandes volumes de dados e a necessidade de se exibir resultados em tempo real, assim como a queda no preço da memória DRAM, surge a necessidade de se manter cada vez mais dados em memória volátil. Nesse cenário, o Projeto RAMCloud propõe um sistema de armazenamento de dados inteiramente em memória DRAM, garantido durabilidade, disponibilidade e baixíssima latência de acesso aos dados.

Sumário

Capítulo 1

Introdução e Contexto

A memória *DRAM*, do inglês *Dynamic Random-access Memory*, é um tipo de memória volátil de acesso aleatório, ou seja, seus dados podem ser acessados em tempo aproximadamente constante independentemente de sua posição na memória. Por ser estruturalmente simples (apenas um transistor e um capacitor por bit) sua densidade é muito maior do que outros tipos de memória (SRAM utiliza até 6 transistores por bit) e, apesar de necessitar que seus capacitores sejam recarregados constantemente para manter os dados, tornou-se assim a memória principal na grande maioria dos computadores.

Nos últimos 15 anos o uso de memória DRAM em sistemas de armazenamento de dados vem aumentando para atender às necessidades de aplicações web de larga escala. Isso ocorre principalmente porque embora a capacidade de armazenamento dos discos rígidos (hard disk ou HD, em inglês) tenha aumentado exponencialmente, tanto sua latência de acesso quanto sua vazão de dados não evoluíram na mesma proporção 1.1. Essa aplicações web lidam necessitam lidar com um grande volume de dados em uma intensidade que um sistema de armazenamento baseado puramente em disco ou memória flash não conseguiriam atender, o que faz com que cada vez mais dados precisem estar disponíveis em memória RAM. Como exemplo, sistemas de busca como o Google ou o Yahoo mantém toda sua estrutura de índices em DRAM.

Embora o uso massivo de memória ram venha crescendo, sua principal utilização para sistemas de armazenamento de dados é na forma de *cache*. Nesse caso a aplicação tem que ficar responsável por manter o cache coerente com a base de dados, o que aumenta sua complexidade e limita o seu desempenho pela quantidade de erros de coerência do cache (*cache misses*).

O projeto RAMCloud é um sistema de armazenamento que propõe manter os dados em DRAM de modo persistente e distribuído. Suas três premissas são: baixa latência, escalabilidade e durabilidade. Quando utilizado em um *cluster* interligado por redes de tecnologia de ponta (por exemplo redes *Infiniband*) atinge a latência de 5μs para leitura e 15μs para escrita. Esse cluster pode ser composto facilmente por 10.000 máquinas com um único sistema de endereçamento de dados do tipo chave-valor, o que implica em uma capacidade inicial de 1PB de DRAM. Para garantir a durabilidade e a disponibilidade os dados também são armazenados em um sistema secundário baseado em disco, de um modo que não degrade o desempenho do sistema e que permita a recuperação em caso de falha em menos de dois segundos.

	Mid-1980's	2009	Change
Disk capacity	30 MB	500 GB	16667x
Max. transfer rate	2 MB/s	100 MB/s	50x
Latency (seek & rotate)	20 ms	10 ms	2x
Capacity/bandwidth (large blocks)	15 s	5000 s	333x
Capacity/bandwidth (1KB blocks)	600 s	58 days	8333x
Jim Gray's rule	5 min	30 hours	360x

Figura 1.1: Comparação entre discos rígidos com 25 anos de diferença. Os números em vermelho indicam o quanto piorou o desempenho

Capítulo 2

O problema da latência

2.1 Por que a latência importa ?

A principal motivação do projeto RAMCloud é criar um sistema de armazenamento de dados com uma latência consideravelmente menor do que os outros sistemas existentes. Tipicamente, aplicações web de larga escala são executadas em muitos servidores dentro de um *datacenter*, o qual possui máquinas separadas para executar o código da aplicação e armazenar os dados 2.1. Quando essa aplicação recebe uma requisição, é necessário acessar dados que estão no servidor de dados, o que leva de 0.5ms a 10ms.

Devido à latência de acesso aos dados, uma aplicação que atenda milhões ou até mesmo bilhões de usuários não consegue processar muitas requisiçoes de dados aleatórios para uma dada requisição do usuário. Por exemplo, para que o Facebook possa ter um tempo de resposta rasoável só é possível que ele faça cerca de 150 requisições de dados por requisição de usuário (ainda assim utilizando servidores de cache), o que acaba por limitar suas funcionalidades.

O propósito do projeto RAMCloud é conseguir prover a menor latência possível para pequenos acessos aleatórios em aplicações web de larga escala. Com uma latência de acesso de cerca de 5μs é uma melhora de 50 a 1000x sobre os sistemas de armazenamento de dados atuais.

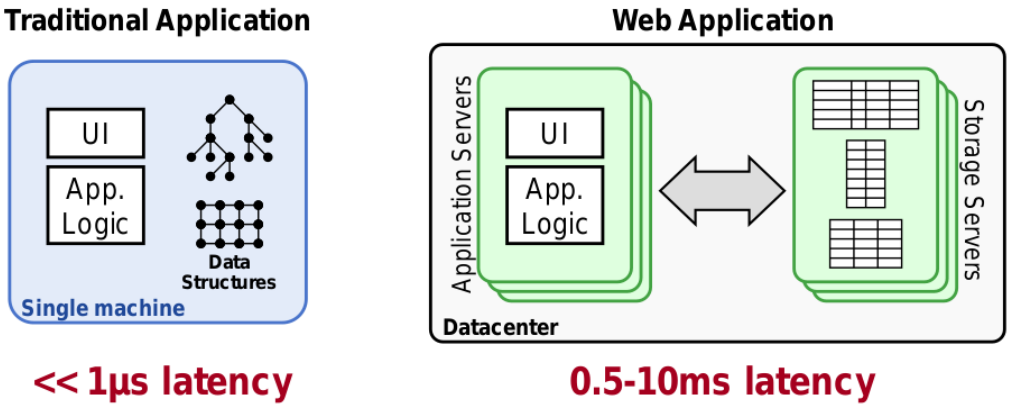

Figura 2.1: Em uma aplicaçao tradicional, os dados e o código estão na mesma máquina (latência de 50-100ns). Já em uma aplicação web escalável, como os dados e o código da aplicação estão em máquinas separadas, a latência pode chegar a até 10ms.

2.2 Atingindo baixa latência

O principal obstaculo para atingir latências da ordem de 5μs a 10μs é a infraestrutura de rede. No início do projeto, no ano de 2009, o tempo típico de uma RPC[1] era de centenas de μs 2.2. Além disso, os prórpios mecanismos do sistema operacional contribuem para aumentar a latência como chamadas ao kernel, a pilha de rede, interrupções, mecanismos de sincronização e o tempo de comunicação entre a CPU e a placa de rede.

[1]RPC - Remote Procedure Call ou Chamada de procedimento remoto é, basicamente, a operação que permite que um processo possa se comunicar ou outro processo em outra máquina geralmente por meio da rede

Component	Traversals	2009	Possible 2014	Limit
Network switches	10	100-300 µs	3-5 µs	0.2 µs
Operating system	4	40-60 µs	0 µs	0 µs
Network interface controller (NIC)	4	8-120 µs	2-4 µs	0.2 µs
Application/server software	3	1-2 µs	1-2 µs	1 µs
Propagation delay	2	1 µs	1 µs	1 µs
Total round-trip latency		**150-400µs**	**7-12 µs**	**2.4 µs**

Figura 2.2: Latência média por componente uma uma RPC. *Traversals*: quantas vezes um pacote precisa passar por cada componente em uma requisição completa (round-trip). *2009*: latência típica de um datacenter em 2009. *Possible 2014*: latência possível de se atingir em 2014 a um custo razoável.

Mesmo com os avanços da tecnologia, a maior parte da latência ainda é gasta na rede ou na placa de rede, o que deixa cerca de 1μs para o sistema processar cada RPC. Sendo assim, para atingir esse objetivo foi preciso desenvolver um sistema no qual:

- Servidores e aplicações precisam enviar e receber pacotes sem passar pelo kernel do sistema operacional;
- Mecanismos de sincronização precisam ser evitados;
- Minimizar os erros de coerência de cache;
- Elimizar os mecanismos de *batching* da placa de rede, onde um certo número de pacotes é acumulado e transmitidos de uma só vez.

No RAMCloud os registradores da placa de rede são mapeados diretamente no espaço de memória da aplicação, permitindo uma comunicação direta entre aplicaçao e placa de rede. Além disso, se uma *thread* está esperando pela resposta de uma RPC, ao invés de ir dormir, ela fica constanstemente verificando a placa de rede. Essa técnica, chamada de *busy waiting* (espera ocupada em português), ajuda a minimizar a latência da troca de contexto do sistema operacional.

Embora a melhor maneira de se minimizar a latência seja utilizar uma única thread para tratar todas as requisições, para garantir a tolerância a falhas os servidores RAMCloud utilizam múltiplas threads: uma única *dispatch thread* e várias *worker threads*. A *dispatch thread* trata de toda a comunicação de rede (requisições e respostas) e também tem a função de selecionar uma

worker thread para processar cada requisição (figura 2.3). Essa comunicação entre threads também é realizada pela técnica de busy waiting.

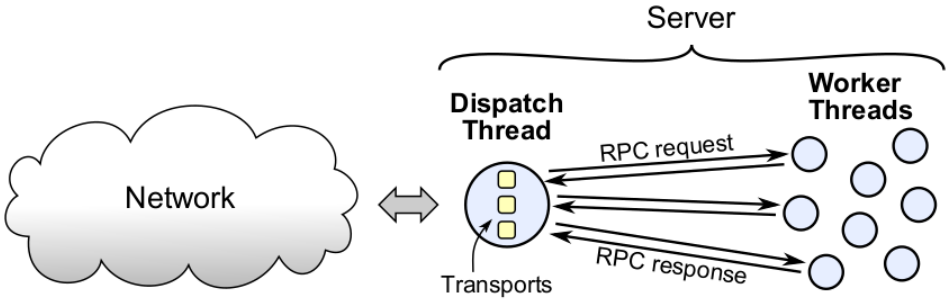

Figura 2.3: Arquitetura de threads do RAMCloud

Capítulo 3

Arquitetura RAMCloud

Manter os dos em DRAM de modo persistente garante que os desenvolvedores de aplicações que utilizem o modelo RAMCloud não precisem gerenciar explicitamente um sistema de backup secundário. O projeto RAMCloud prove esse mecanismo por meio de um modelo de dados chave-valor, organizado na memória DRAM no formato de *log* e, também por meio de uma arquitetura específica que viabilize suas premissas (Veja o Cap. 1).

3.1 Arquitetura dos servidores

O projeto RAMCloud é um software que roda em um cluster de servidores comuns (*commodity servers*). Esse cluster é composto de uma coleção de servidores de armazenamento e um coordenador (figura 3.1). Cada servidor de armazenamento é dividido em duas componente principais: a componente principal ou *master* é responsável por atender às requisições de leitura e escrita; a componente secundária ou *backup* é responsável por manter em disco uma cópia dos dados de outras componentes principais de outros servidores.

Os dados dentro do RAMCloud são organizados em tabelas que, por sua vez, são segmentadas em estruturas menores denominadas *tablets*. Uma tabela grade, composta por muitas *tablets*, pode estar espalhada em vários servidores, idealmente de maneira uniforme.

O coordenador é responsável por gerenciar o cluster, ou seja, saber quais são e onde estão os servidores e tabelas. Embora exista apenas um coordenador ativo, para garantir tolerância a falhas, pode haver um ou mais coordenadores de prontidão para assumir o controle em caso de falha. Para

que não ocorra gargalos, o coordenador não pode estar envolvido em operações de dados como leitura e escrita, o qual apenas informa à aplicação quais tabelas estão em cada servidor para permitir o acesso diretamente.

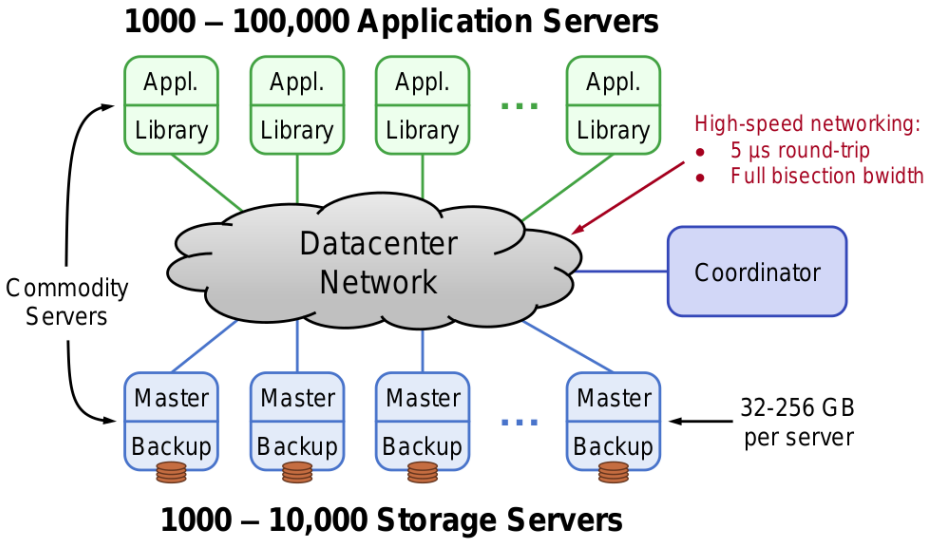

Figura 3.1: Cluster RAMCloud

3.2 Modelo de dados

O modelo de dados do RAMCloud é o chave-valor com algumas extensões para garantir o desempenho e a escalabilidade. Por exemplo, não é possível atribuir uma chave sequencial para cada elemento em uma tabela. Isso forçaria todas as inserções a passarem por um mesmo servidor, limitando a escalabilidade do sistema.

Cada tabela dentro do RAMCloud é identificada univocamente por um nome e um identificador de 64-bit. Os objetos de cada tabela contém uma chave variável de 64KB, única em uma dada tabela, um valor de tamanho variável e até 1MB e um número de versão de 64-bit. Um objeto que foi sobrescrito sempre terá um novo número de versão maior do que todos os outros anteriores para esse mesmo objeto.

As principais operações sobre os dados no RAMCloud são:

- *createTable(name) -> id*: cria uma nova tabela com o nome dado e retorna seu identificador;

- *getTableId(name) -> id*: retorna o identificador da tabela identificada por "name";
- *dropTable(name) -> id*: remove a tabela identificada por "name";
- *read(tableId, key) -> value, version*: retorna o valor e a versão do objeto identificado por "key"na tabela identificada por "tableId";
- *write(tableId, key, value) -> version*: cria um ou sobrescreve o objeto identificado por "tableId","key";
- *delete(tableId, key)*: remove um objeto;
- *conditionalWrite(tableId, key, value, condition) -> version*: operação de escrita condicional. Por exemplo, a condição pode exigir que um objeto tenha uma determinada versão ou não exista previamente;

3.3 Armazenamento

A estratégia de armazenamento do RAMCloud é manter uma cópia dos dados em DRAM e ao menos três copias em disco de servidores distintos. Esses dados são armazenados em uma estrutura de *log* similar a um sistema de arquivos em logs[1]

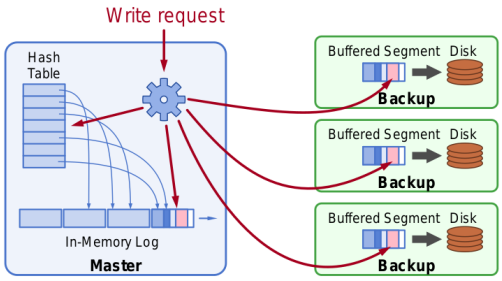

Figura 3.2: Ao receber uma requisição de escrita, é criada uma entrada na tabela de hash do servidor master e o objeto escrito em DRAM. Simultaneamente esse mesmo objeto é escrito na DRAM de ao menos três servidores backup e posteriormente escrito em disco. A escrita retorna assim q o objeto é escrito na DRAM dos servidores backup, minimizando a latência

[1]LFS - Log-structured File System, Rosenblum and Ousterhout 1992

3.4 Recuperação de falhas

Para recuperar um servidor master com problemas em menos de 2 segundos, os dados de backup precisam estar espalhados em diversos nós e a tarefa ser executada de forma paralela. Essa tarefa de recuperação é executada pelo coordenador.
Ao perceber que um servidor "caiu", o coordenador seleciona alguns servidores para recuperar os dados perdidos, denominados *recovery masters*. Cada *recovery master* recebe um subconjunto de *tablets* das tabelas que estavam no servidor com problema, que então incorpora esses objetos às suas entradas de hash, tornando-se o novo hospedeiro desses dados. A figura 3.3 ilustra o processo.

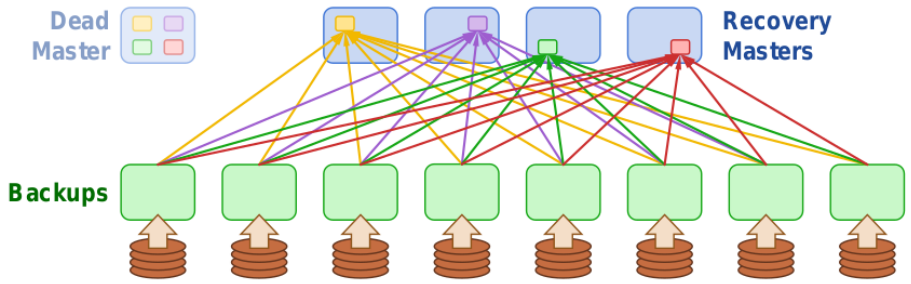

Figura 3.3: Recuperação de um servidor master

Capítulo 4

Conclusões

Por mais que o projeto RAMCloud já tenha uma versão que os desenvolvedores dizem ser utilizável, algumas limitações, deficiências e a falta de uma aplicação que realmente possa tirar vantagem dessa arquitetura podem acabar retardando ou mesmo inviabilizando sua utilização atualmente. O sistema em geral depende fortemente do desempenho da rede de intercomunicação, nada menos do que a tecnologia de ponta que existe hoje, o que pode implicar em alto custo de implantação e eventuais falhas de hardware, software ou mesmo de projeto dessas tecnologias. Além disso, para que o RAMCloud funcione como previsto, todos os servidores precisam estar no mesmo datacenter uma vez que está implícito que os servidores podem se comunicar com baixa latência.

Do ponto de vista de armazenamento, para permitir que o sistema se recupere de falhas em menos de dois segundos, cada servidor de backup não pode armazenar mais do que cerca de 500MB de log, ou seja, em um cluster com 10 nós cada servidor master pode armazenar no máximo 5GB de DRAM. Além disso, o modelo de dados chave-valor adotado é bem simples e eficiente para acesso, mas aplicações que necessitam de um modelo de dados mais estruturado, com índices secundários e transações de múltiplos objetos (ou mesmo um modelo ACID) teriam que ser totalmente reescritas para funcionar no RAMCloud. Falta também algum sistema de proteção de acesso aos dados, atualmente qualquer cliente pode modificar qualquer dado armazenado.

Talvez ainda não exista um nicho específico para o projeto RAMCloud embora não deixe de ser um projeto interessante e bastante promissor, basta analisar o mercado de software para chegar a conclusão de que existe uma

demanda cada vez maior, por parte das aplicações web, para manter dados em memória RAM. Tecnologias oriundas da memcomputação, principalmente os memristores, podem viabilizar a criação de um novo tipo de memória que agregue baixa latência, velocidade superior as DRAM e capacidade de um disco rígido, tornando não só o RAMCloud como diversas tecnologias atuais obsoletas.

One machine can do the work of fifty ordinary men.
No machine can do the work of one extraordinary man.
–Elbert Hubbard

Referências Bibliográficas

[1] John Ousterhout, The RAMCloud Project, 2015
https://ramcloud.atlassian.net/wiki/display/RAM/RAMCloud

[2] John Ousterhout, The RAMCloud Storage System, 2014
https://ramcloud.atlassian.net/wiki/download/attachments/6848571/RAMCloudPaper.pdf

[3] John Ousterhout, The Case for RAMCloud, 2011
http://portal.acm.org/citation.cfm?id=1965751

[4] John Ousterhout, A collection of all of John Ousterhout's RAMCloud slides, 2012
https://ramcloud.atlassian.net/wiki/display/RAM/RAMCloud+Presentations

[5] John Ousterhout, Introductory talk on RAMCloud, 2011
http://www.youtube.com/watch?v=lcUvU3b5co8