

MIT BIG DATA

Bloco Armazenamento Heterogeneo de Dados

Título: VoltDB

Grupo: Arvin Shahidi

Ciro Andrade

Natassja Ferraz

Rodrigo Hartmann

Professor: Eduardo Morelli

Rio de Janeiro, Junho de 2016

Sumário

[**Detalhamento Teórico** 2](#_Toc455586812)

[**1.** **Aplicações** 2](#_Toc455586813)

[**2.** **Arquitetura** 2](#_Toc455586814)

[**3.** **Instalação** 4](#_Toc455586815)

[**4.** **Interação** 5](#_Toc455586816)

[**5.** **Tarefas Administrativas** 5](#_Toc455586817)

[**Evidências** 6](#_Toc455586818)

[**Conclusões** 7](#_Toc455586819)

**Detalhamento Teórico**

1. **Aplicações**

Aplicações que tenham como requisito transações ACID. Por se tratar de um banco de dados New SQL o VoltDB se apresenta como uma opção de banco transações ACID entregando performance de I/O.

1. **Arquitetura**

**Memória Page Buffer Management:** O sistema de paginação de buffer atribui dados em páginas de tamanhos fixos, organiza a sua colocação dentro das páginas e depois faz a administração das páginas carregadas na memória e das que estão no disco. As páginas devem ser rastreadas através de indicadores como ‘boa’, ‘ruim’, ‘lida’ e ‘escrita’. Todo esse subsistema existe para gerenciar o armazenamento em disco, ou seja, esse subsistema não agrega valor a um sistema ‘in-memory’.

**Concurrency Management:** O banco de dados resolve dois problemas importantes de concorrência. Primeiro, existe o problema lógico que várias transações do usuário operando simultaneamente não devem ter conflitos e devem ler dados consistentes. Isso é feito usando “locks” de alto nível sobre os recursos, como tabelas e linhas, e acompanhando esses “locks” em toda a transação. Em contrapartida, as bases de dados atuais são executadas em ‘multi trheads’, precisam ser ‘thread-safe’, e ter baixo custo de execução.

1. ***Escalonamento Horizontal:*** Além da memória, a segunda grande mudança é a mudança para sistemas de escalonamento horizontal. Ultimamente, a lei de Moore tem sido usada para nos dar mais núcleos, ao invés de um desempenho mais rápido single-core. ‘Hardware Clustering’ como commodity é realidade e hardware de tolerância a falhas é a aposta da mesa. Escala horizontal tem sido muito difícil para tradicionais RDBMSs. Muitos têm opinião de que o modelo relacional não é escalável. O sistema C-Store mostrou que o modelo relacional pode ser escalado para SQL analíticos.
2. ***One Thread*** - Uma tarefa de cada vez: Há muitas maneiras de tornar o custo de concorrência menor, mas poucos conhecem as necessidades do analista de dados. Todas as operações de dados em VoltDB são single-threaded, executando cada operação até o fim antes de iniciar a próxima. Estruturas de dados ‘Fancy concurrent B-Trees’ e ‘Lockless’ foram substituídas por estruturas simples sem segurança ‘no thread’ e sem custo de acesso simultâneo. (Nota-se que esta escolha só é realmente possível com um projeto de memória-centric. A latência de leitura e escrita do disco é frequentemente escondida por memória compartilhada multi-threading, mas ao executar single-threaded, que a latência fará com que o CPU passar a maior parte do seu tempo ocioso. Eliminando disk waits permitiu manter o CPU saturado, e executar anéis em torno de arquiteturas tradicionais.)
3. ***No Waiting on Users:*** Como um banco de dados operacional, as "operações" VoltDB em questão são inteiramente transações ACID na verdade, com várias fases de leitura, escrita e lógica condicional. Se o sistema vai executar operações até o final, uma após a outra, a latência do disco não é o único ponto que deve ser eliminado, também é necessário eliminar a espera nas transações do usuário. Existem alguns inconvenientes aqui, o controle de transação externo pode trazer certos problemas com mais facilidade.
4. ***Concurrency through Scheduling, Not Shared Memory:*** O VoltDB resolveu o problema de concorrência ao executar uma coisa de cada vez, usando um pipeline single-threaded. Para utilizar processamento multi-core, por que não simplesmente mover a abstração? Trate quatro máquinas com quatro núcleos cada, como dezesseis máquinas. Cada núcleo recebe um pipeline single-threaded onde o sistema é montado para geri-los de forma agregada, e assim o banco de dados pode ser escalado para lotes de núcleos, lotes de nós ou ambos. Neste modelo não há estruturas de dados simultâneos, locks ou outras sincronizações pesadas na camada do gerenciamento de dados. Para manter os pipelines completos, VoltDB vai particionando os dados. Digamos que um aplicativo de finanças é dividido pelo símbolo ticker; agora todas as operações em "MSFT" podem ser dirigidas para o pipeline apropriado. Se um aplicativo é dividido pelo cliente, uma operação em dados de um cliente específico pode ser encaminhada facilmente ao pipeline correto. Também é possível operar os dados através de pipelines, quebrando o trabalho em pedaços por pipeline e fazendo o processamento de resultados por pipeline em resultados globais. VoltDB chama isso de "simultaneidade através de agendamento." Nesta abordagem ainda há linhas e locks no sistema, principalmente para apoiar as operações de redes e programação, mas toda a lógica de SQL e procedimento de Java é single-threaded e extremamente rápido. Os locks estão lá para agendar coisas que são de baixa contenção; eles protegem pequenas estruturas de dados agendadas, ao invés de dados do usuário para uma transação.
5. ***Modern Disk Persistence:*** VoltDB tem necessidade de persistência em disco para estar pronto para o mercado empresarial. Para começar, VoltDB mal lê a partir do disco, assim grande parte da carga de trabalho de um sistema tradicional é removida. Em segundo lugar, o IO de disco do VoltDB é quase 100% de anexação só de escritas streaming. Mesmo discos giratórios podem sustentar alta taxa de transferência de gravação, quando usado desta forma. Em terceiro lugar, IO de disco é quase totalmente paralela à carga de trabalho operacional. O sistema é projetado para quase nunca bloquear a sincronização de disco. Isto é conseguido através de dois mecanismos: background snapshots e inter-snapshot logical logging. VoltDB background snapshots é transacional e serializa os dados em disco em um único, ‘logical point-in-time’, ‘cluster-wide’. Eles prosseguem na velocidade do disco; um disco mais lento terá menos snapshots por hora. Eles também não bloqueiam o trabalho operacional em curso. Um registro lógico de todas as operações de gravação é transmitido para o disco. Se um cluster inteiro falhar, o snapshot mais recente é recarregado, seguido por uma repetição do log lógico para trazer o cluster de volta para o ponto de falha. Este log lógico tem uma enorme vantagem sobre o rendimento de logs binários, em parte porque ela é limitada em tamanho, mas principalmente porque IO de disco pode começar antes do trabalho operacional ser iniciado. Combinado com um grupo de “commit mechanism”, o registo (escrita) do VoltDB é notavelmente de baixo impacto, permitindo milhões de gravações por segundo, por nó com persistência disco síncrona.
6. ***High Availability & Leveraging Determinism:*** Enquanto persistência em disco é grande, em um paralelo, em um cluster shared-nothing, é imperativo manter os dados em mais de um lugar. A alta disponibilidade é a promessa que VoltDB continuará a ser executado em face de muitas falhas de hardware e de rede comum. Através da replicação, VoltDB alcança alta disponibilidade e segurança de dados adicionais. O método típico de replicação em clusters shared-nothing é a utilização de um nó mestre e um ou mais nós de réplica. As alterações são aplicadas pela primeira vez para o mestre, em seguida, é enviada de forma síncrona ou assíncrona para a(s) réplica(s). Esta abordagem tem várias desvantagens: Se a replicação é síncrona, os custos de latência adicionais são adicionados por replicação. Se a replicação é assíncrona, os dados podem ser perdidos quando o mestre falhar. VoltDB usa replicação síncrona, sem a sobrecarga de latência. Ele faz isso por ser um pouco não convencional. Já que todas as operações (transações) executadas no VoltDB são auto-suficientes e deterministas, é suficiente simplesmente replicar uma lista ordenada de operações à serem executadas, e deixar que cada réplica faça o mesmo trabalho. Todas as réplicas fazendo o mesmo trabalho na mesma ordem, ao mesmo tempo em paralelo. As respostas são enviadas para o cliente quando todas as réplicas confirmam a conclusão. VoltDB garante que as operações são deterministas e estado da réplica é idêntica em várias formas, incluindo hash distribuída de mutação de dados. Replica mastership implica apenas uma réplica dentro de um conjunto de réplicas é responsável por ordenar as gravações deterministas. A falha do nó irá desencadear uma detecção de falhas e um mecanismo de resolução que re-elege mestres para todos os conjuntos de réplicas.
7. **Instalação**

***Sistema Operacional e Requisitos de Software***

* Sistema Operacional

VoltDB requires a 64-bit Linux-based operating system. Kits are built and qualified on the following platforms: CentOS version 6.6 or later, including 7.0 Red Hat (RHEL) version 6.6 or later, including 7.0 Ubuntu versions 12.04 and 14.04 Development builds are also available for Macintosh OS X 10.9 and later.

* CPU

Dual core x86\_64 processor 64 bit 1.6 GHz

* Memoria

4 Gbytes

* Java

VoltDB Server: Java 8 Java and JDBC Client: Java 7 or 8

* Requisitos de Software

NTP5 Python 2.6 or later release of 2.x

* Recomendação de Software

Eclipse 3.x (or other Java IDE)

***Instalação***

VoltDB é distribuído como um arquivo tar comprimido para cada uma das plataformas suportadas. O nome do arquivo identifica a plataforma, a edição (comunidade ou empresa) eo número da versão. O melhor caminho para instalar VoltDB é para descompactar o kit de distribuição como uma pasta no diretório inicial da sua conta pessoal, igual a

$ tar -zxvf voltdb-ent-6.0.tar.gz -C $HOME/

Instalando em seu diretório pessoal dá-lhe acesso total ao software e é mais útil para o desenvolvimento. Se você estiver instalando VoltDB em um servidor de produção onde o banco de dados será executado, você pode querer instalar o software em um local padrão do sistema para que o cluster de banco de dados pode ser iniciado com os mesmos comandos em todos os nós. Os seguintes comandos shell instalar o software VoltDB na pasta

/opt/voltdb:

$ sudo tar -zxvf voltdb-ent-6.0.tar.gz -C /opt

$ cd /opt

$ sudo mv voltdb-ent-6.0 voltdb

1. **Interação**
2. **Tarefas Administrativas**

* Inicializar um banco de dados VoltDB

$ voltdb create

* Inicializar o banco de dados em um cluster

$ voltdb create --deployment=deployment.xml -–host=voltsvr1

Ou você também pode usar formas mais curtas para as bandeiras de argumento:

$ voltdb create -d deployment.xml -H voltsvr1

* Atualizando nós no cluster

Use the voltdb add command to start up and add a new node to the running database cluster. See

Section 9.2.1, “Adding Nodes with Elastic Scaling”.

Use the voltdb rejoin command to restart a node that was previously part of the cluster but had stopped

running. See Section 10.3, “Recovering from System Failures”.

* Parando um banco de dados VoltDB

$ voltadmin shutdown

* Reiniciando um banco de dados VoltDB

$ voltdb recover

$ voltdb recover -d deployment.xml -H voltsvr1

$ voltdb recover -d deployment.xml -H voltsvr1 \

* Definindo a configuração do cluster

<?xml version="1.0"?>

<deployment>

<cluster hostcount="n"

kfactor="n"

/>

</deployment>

* Determinar quantos sites por host

<?xml version="1.0"?>

<deployment>

<cluster . . .

sitesperhost="6"

/>

</deployment>

* Configurando caminhos para recursos de tempo de execução

<paths>

<voltdbroot path="/opt/voltdb" />

</paths>

You can also identify specific path locations for individual features including:

• <commandlog>

• <commandlogsnapshot>

• <exportoverflow>

• <snapshots>

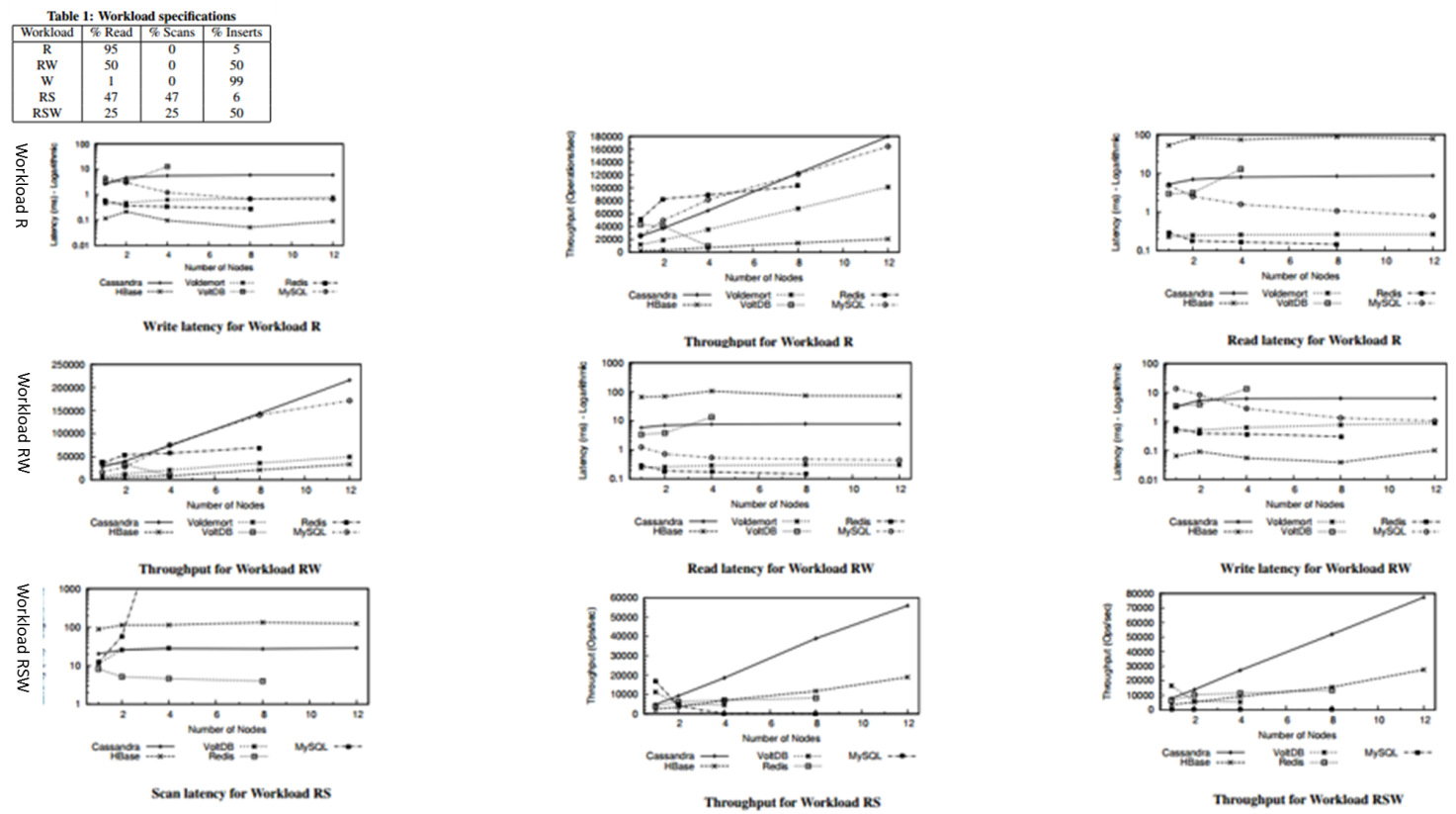
<paths>

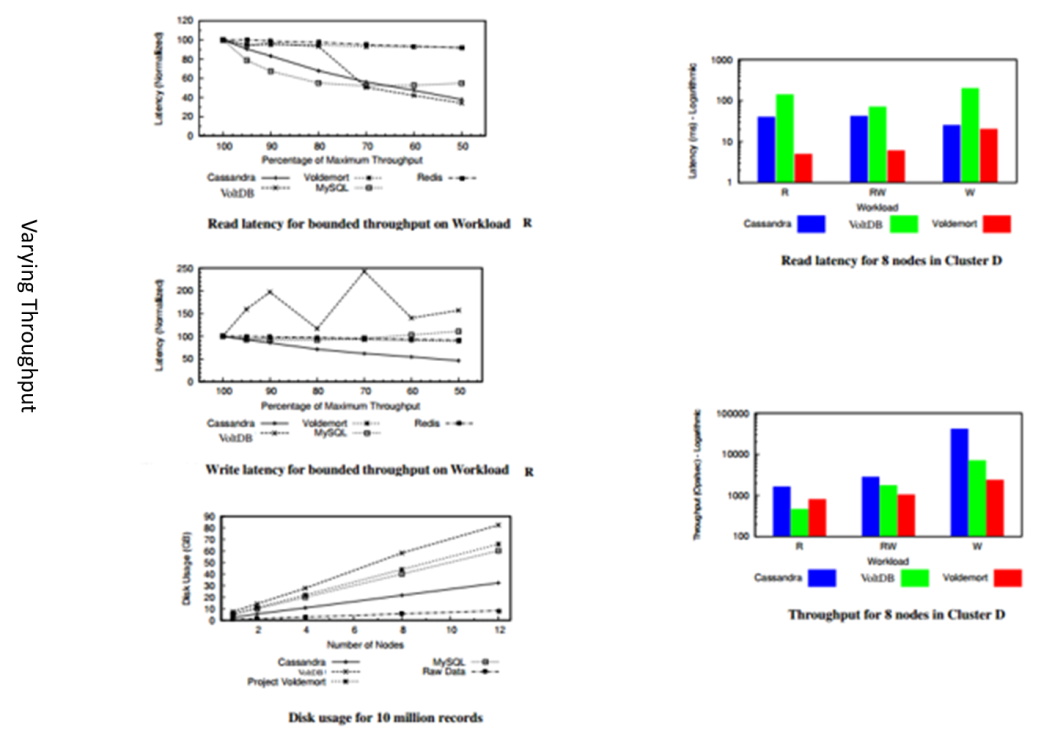
<voltdbroot path="/opt/voltdb" />

<exportoverflow path="/opt/overflow" />

</paths>

**Evidências**

****

****

**Conclusões**

O VoltDB se apresenta como uma boa solução em casos específicos de muitos usuários acessando a mesma aplicação gerando escrita ‘fina’ e constante, o caso que explicita essa vantagem é utilizar o VoltDB em um sistema de CDR (telecom) como uma camada anterior a uma camada de SLQ padrão aonde serão realizadas as leituras.