Instituto Infnet

MIT em BIG DATA

Bloco C – Armazenamento Heterogêneo de Dados

Projeto de Bloco

Banco de Dados - VOLTDB

Setembro/2016

Alunos:

Maria Fátima Soares de Souza

Lucas Pereira de Sousa

Professor:

Eduardo Morelli

Índice

Introdução

O presente documento tem por objetivo apresentar o trabalho para o Bloco C – Armazenamento Heterogêneo de Dados do curso MIT – Big Data do Instituto Infnet. O assunto que será apresentado é o banco de dados VoltDB.

O VoltDB é um banco de dados relacional que vem atingir um ramo onde os RDBMS (*RelationalDatabase Management System*) estavam perdendo mercado para os NoSQL, que é o mercado de escalabilidade.Ele é um banco de dados "em memória" desenhado por Michael Stonebraker (que esteve envolvido no Ingres e POSTGRES), Sam Madden, e Daniel.

A primeira versão é de 2010 e atualmente está na versão 6.

1) Visão Geral

1.1) O que é o VoltDB

Banco de dados relacionalprojetado para aplicações críticas de negócios de alto desempenho. Sua arquitetura permite adicionar facilmente processadores no cluster para atender ao crescimento do volume de dados e de transações.

O VoltDB utiliza o armazenamento em memória para maximizar a produtividade, evitando o acesso ao disco.

Ganhos adicionais de desempenho são obtidos através da serialização de todos os acessos aos dados, evitando assim muitas das demoradas funções dos tradicionais banco de dados tais como: bloqueio, travamento e a manutenção de transações de log.

A escalabilidade, confiabilidade e alta disponibilidade são alcançadas através de *clustering* e replicação de múltiplos servidores e *farms* de servidores.

O VoltDB é um banco de dados transacional totalmente *ACID* (Atomicidade, Consistência, Isolamento e Durabilidade).Além disto, por usar o ANSI SQL padrão para a definição dos esquemas e do acesso aos dados, ele também é facilmente assimilado pelos designers de banco de dados como pelos desenvolvedores.

1.2) Quem deve usar o VoltDB

O VoltDB não se destina a resolver todos os problemas de banco de dados. É destinado a um segmento específico de computação empresarial. Ele é voltado especificamente para *dados rápidos*. Ou seja, aplicativos que devem

processar grandes fluxos de dados rapidamente. Isso inclui aplicações financeiras, aplicações de mídia social, e o c*rescente* campo da Internet das Coisas. Os principais requisitos para estas aplicações são escalabilidade, confiabilidade, alta disponibilidade e rendimento excepcional.

VoltDB é utilizado hoje para as aplicações tradicionais de alto desempenho, tais como: alimentações de dados do mercado de capitais, comércio financeiro, fluxo de sensores *Telco* e sistema de distribuição baseado em sensores. Ele também é utilizado em aplicações *emergentes* tais como: wireless, games online, detenção de fraude, trocas de anúncios digitais, etc. Qualquer aplicação que necessite de alta taxa de transferência de dados, escalonamento linear e precisão de dados.

Entretanto o VoltDB não é indicado para todos os tipos de consulta. Por exemplo, ele não é a escolha ideal para o acesso e comparação de grandes conjuntos de dados históricos que precisam ser consultados em várias tabelas. Este tipo de atividade é comumente encontrada em soluções *BI (Business Intelligence)* e DW (*Data Warehousing)* onde já existem banco de dados mais adequados.

Para ajudar as empresas que exigem tanto o desempenho excepcional da transação como os relatórios Ad Hoc, o VoltDB inclui funções integradas para que os dados históricos possam ser exportados para um banco de dados analítico para mineração de dados em maior escala.

1.3) Como o VoltDb funciona

VoltDB não é como os produtos de banco de dados tradicionais. Cada banco de dados VoltDB é otimizado para uma aplicação específica dividindo as tabelas de banco de dados e as *stored procedures* que acessam essas tabelas através de múltiplos "*sites*" ou partições em uma ou mais máquinas host para criar o banco de dados distribuído.
Como tanto os dados e o trabalho são divididos, várias consultas podem ser executados em paralelo, ao mesmo tempo, porque cada local opera de forma independente. Cada transação pode ser executadae concluída sem a interferência de bloqueio individuais de registros que consome a maior parte do tempo de processamento nas bases de dados tradicionais.
Finalmente, VoltDB balanceia os requisitos de máxima performance com a flexibilidade para acomodar as menos intensas, mas igualmente importantesconsultas que atravessam as particições.

1.3.1)Particionamento de Tabelas

Em VoltDB, cada procedimento armazenado, *stored procedure*, é definido como uma transação. A *stored procedure* (i.e. transação) sucedeou reverte como um todo, garantindo a consistência do banco de dados.

Ao analisar e pré-compilar a lógica de acesso a dados nos procedimentos armazenados (stored procedures), VoltDB pode distribuir tantoos dados eo processamento associado com ele para as partições individuais no cluster. Desta forma, cadapartição contém uma "fatia" única dos dados e do processamento de dados. Cada nó no cluster pode suportarvárias partições.

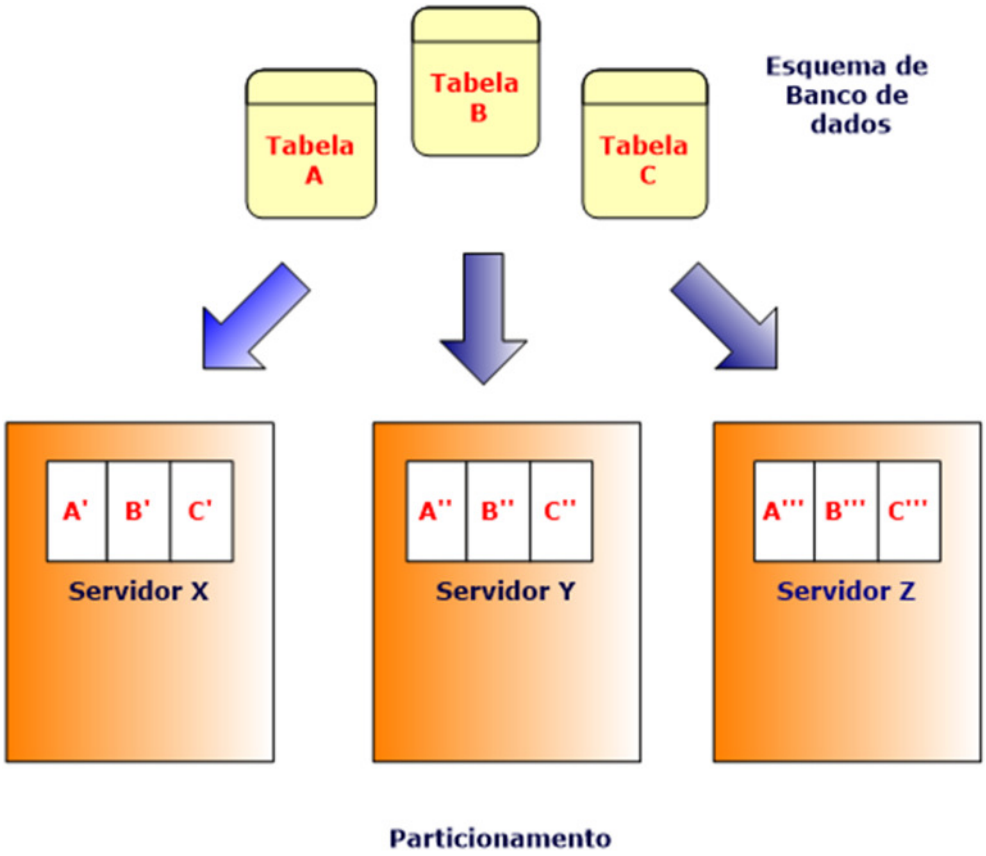
O particionamento organiza o conteúdo de uma tabela de banco de dados em unidades autónomas separadas, semelhante a sharding. O particionamento no VoltDB é único porque:

- O VoltDB particiona automaticamnete as tabelas de banco de dados com base em uma coluna de particionamento especificada. Não se tem que gerir manualmente as partições.

- Pode-se ter várias partições ou sites, em um único servidor. Em outras palavras, o particionamento não é apenas para dimensionamento do volume de dados, mas para o desempenho também.

- O VoltDb particiona os dados e a stored procedure e assim aproveita o paralelismo e obtém ganhos de performance.

Figura 1 – Particionamento de Tabelas



1.3.2)Processo Serializado (single-threaded)

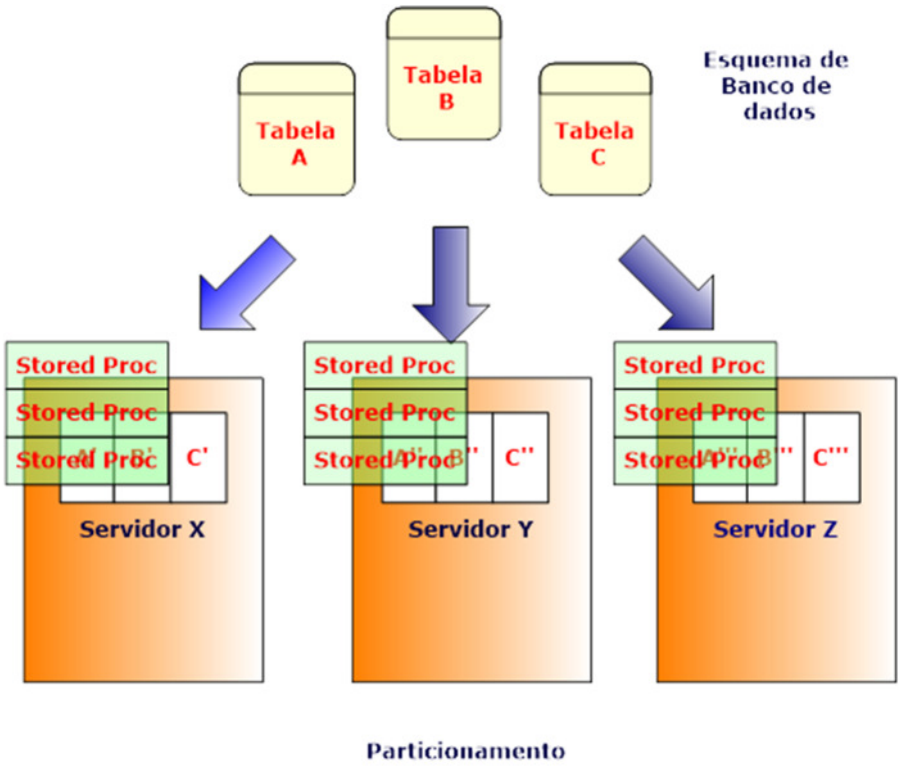
Em tempo de execução, as chamadas para os procedimentos armazenados (stored procedures) são passadas para a partição específica. Quando o procedimento é "Single Partitioning" ou Partição Única (o que significa que operam em dados dentro de uma única partição) ele é executado por um processo do servidor, liberando o restante do cluster para lidar com outras solicitações em paralelo.

Ao usar o processamento serializado, VoltDB garante a consistência transacional sem a sobrecarga de bloqueio, travamento e logs de transações, enquanto o particionamento permite que o banco de dados lide com vários pedidos ao mesmo tempo.

Como regra geral, quanto mais processadores (e portanto mais partições) no cluster, mais transações por segundo são executadas pelo VoltDB.

Quando um procedimento exige dados de várias partições, Multi-partição (Multi-partitioning) um nó age como um coordenador e administra o trabalho necessário com os outros nós, coleta os resultados e completa a tarefa. Esta coordenação faz operações de multi-particionado ligeiramente mais lentas do que as transações Partição Única. No entanto, a integridade transacional é mantida e bem como um rendimento máximo.

Figura - Processamento Serializado



É importante notar que a arquitectura VoltDB é otimizada para minimizar a latência. A latência de qualquer transação (o tempo de quando a transação começa até o processamento termina) é similar em VoltDB para outros bancos de dados.

No entanto, o número de transações que pode ser concluído em um segundo (ou seja rendimento) é mais elevado porque VoltDB reduz a quantidade de tempo que os pedidos se encontram na fila de espera para ser executado.

1.3.3) Tabelas vs. replicado particionado

Tabelas são divididas em VoltDB baseada em uma coluna que o desenvolvedor ou designer especifique.

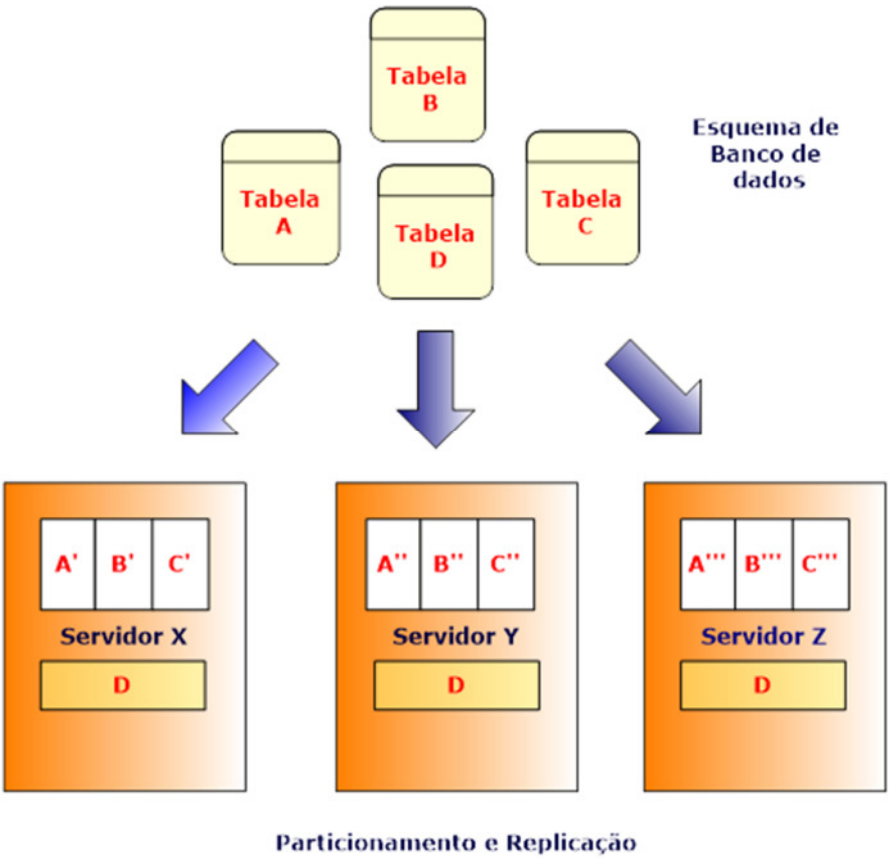
Quando é escolhido o particionamento pelas colunas que correspondem a forma como os dados são acessados pelas *stored procedures*, então a execução é otimizada.

Para otimizar ainda mais o desempenho, VoltDB permite certas tabelas de banco de dados serem replicadas para todas as partições do cluster.

Esta facilidade permite que as pequenas tabelas, que são tipicamente somente leitura, fiquem próximas as tabelas maiores e assim o *join* (junçao) entre elas funcionem como uma operação de *single-partitioning*.

Uma última ressalva relativa a tabelas replicadas: os benefícios de ter os dados replicados em todas as partições é que pode ser lido a partir de qualquer partição individual. No entanto, o déficit é que quaisquer atualizações ou inserções a uma tabela replicada deve ser executada em todas as partições de uma vez. Este tipo de procedimento de várias partições reduz os benefícios de processamento e impacta as transferências paralelas. É por isso que não se deve replicar tabelas que são atualizadas com freqüência.

Figura – Replicação de Tabelas



1.3.4) Facilidade de dimensionamento para atender às necessidades de aplicação

A arquitetura VoltDB é projetada para simplificar o processo de escalar o banco de dados para atender as necessidades de mudança do aplicativo.

Aumentar o número de nós em um cluster VoltDB tanto aumenta a taxa de transferência (aumentando o número de filas simultâneas em operação) como aumenta a capacidade de dados (aumentando o número de partições utilizadas para cada tabela).

Aumentar um banco de dados VoltDB é um processo simples que não requer quaisquer alterações no esquema de banco de dados ou código de aplicação.

1.4) Trabalhando com VoltDB Efetivamente

É possível usar VoltDB como qualquer outro banco de dados SQL, criar tabelas e executar consultas ad hoc usando instruções SQL padrão. No entanto, para tirar o máximo partido das capacidades do VoltDB, é melhor projetar seu esquema e as *stored procedures* para maximizar o uso do particionamento de tabelas e *procedures*.

1.5) Propriedades ACID

Um Banco de Dados ACID é fundamental para qualquer aplicação crítica.

Uma das considerações mais relevantes ao decidir pela utilização ou não de uma tecnologia de banco de dados é saber se o banco é ACID (Atomicity, Consistency, Isolation, Durability), essas características possibilitam garantias que impactam diretamente o negócio.

Atomicidade: Controle sobre início e fim da transação, é a garantia que todo o bloco de transações foi executado integralmente.

Consistência: A garantia de que um dado está íntegro durante e após a transação.

Isolado: Controle sobre os dados de uma transação onde uma transação no banco de dados não pode impactar nos dados das transações em paralelo.

Durabilidade: Controle da persistência do dado garantindo que após o "commit" é necessário que o dado esteja 100% íntegro e disponível mesmo em caso de falha.

O VoltDB é um banco de dados ACID. No que se refere a durabilidade ele possui as seguintes funções:

- K-Safety: Replicação de dados. O K-1 significa que cada partição é replicada em duas máquinas. K-2, signiifca que é replicada em 3 máquinas e assim por diante.

- Snapshot: Cópia das informações num determinado momento. Backup, que pode ser pode ser executado de forma discreta ou contínua (por exemplo, a cada 250 ms)

- Command Logging: De tempos em tempos as informações que estão na memória são armazenadas no disco. Se houver uma falha e o banco tiver que ser restaurado, a base é restaurada e são aplicados os logs do ponto do último backup até o ponto da falha.

Figura – Command Logging

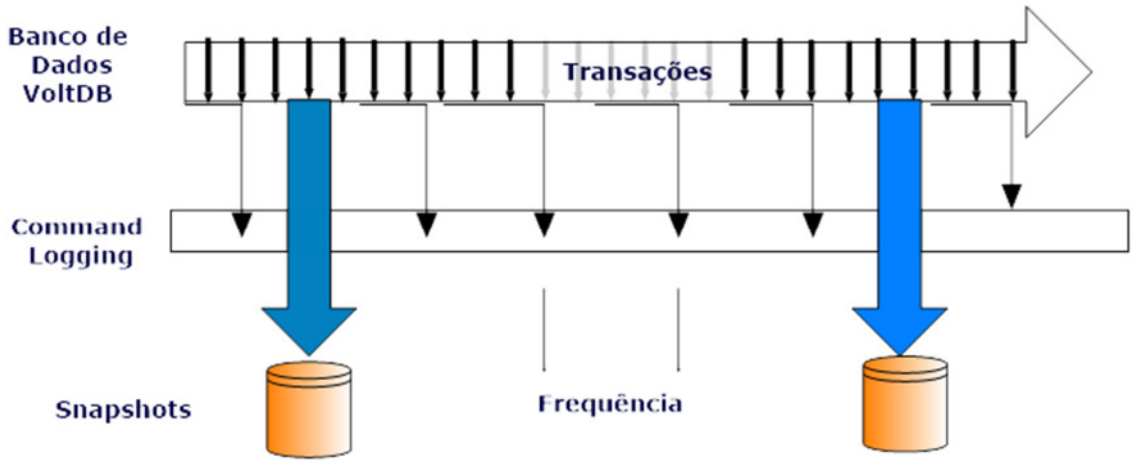
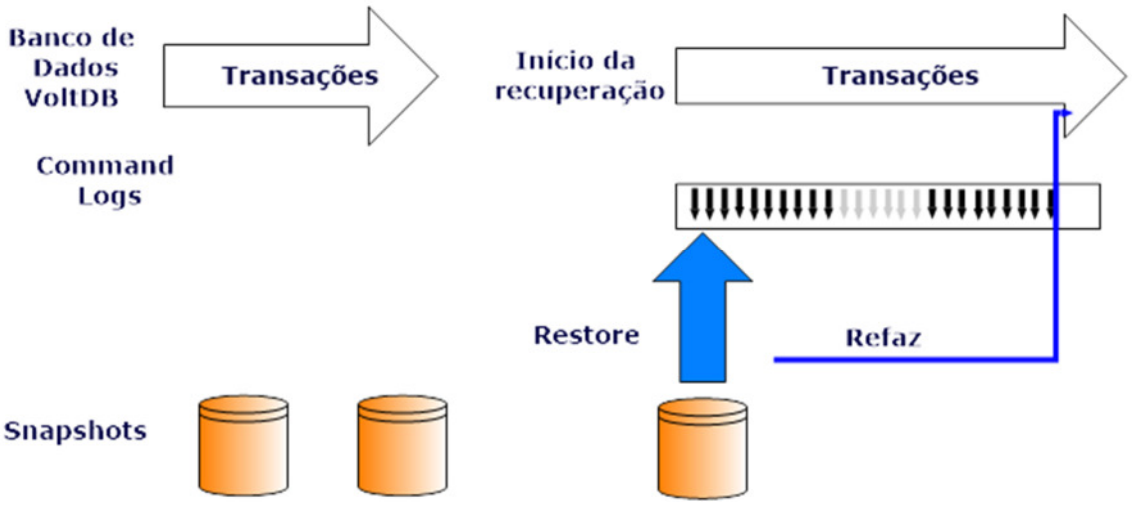


Figura – Command Logging na recuperação



2) Informações Técnicas

Website	Voltdb.com
Desenvolvedor	Voltdb Inc.
Logotipo	
Versão inicial	2010
Versão atual	6.0 de janeiro de 2016
Licença	Open source
Linguagem de programação suportadas	C# , C++, Java, PHP, Python
Método de Particionamento	Sharding
Método de Replicação	Master-slave replicação
Sistema Operacional	Requer Linux 64 bits . Kits são construídos e qualificados para as seguintes plataformas: -CentOS versão 6.6 ou posterior maior, incluindo a 7.0; - RedHat (RHEL) versão 6.6 ou posterior, incluindo a 7.0; - Ubuntu versões 12.04 e 14.04 - Também disponível para Macintosh OS X10.9 ou posterior. **(a)**
CPU	Dual core **(b)** x86_64 processadores 64 bits 1.6 GHz
Memomry	4 Gbytes**(c)**
Java **(d)**	Servidor VoltDb: Java 8 Java and JDBC Client: Java 7 ou 8
Required software	NTP **(e)** Python 2.6 ou versão posterior de 2.x
Software recomendado	Eclipse 3.x (ou outro Java IDE)
(a)	CentOS 6.6, CentOS 7.0, RHEL 6.6, RHEL 7.0 e Ubuntu 12.04 e 14.04 são os únicos oficialmente sistemas operacionais suportados para VoltDB. No entanto, VoltDB é testado em vários outros compatível com POSIX e Linux baseado em sistemas operacionais de 64 bits, incluindo Macintosh OS X 10.9.
(b)	Processadores dual core são um requisito mínimo. Quatro ou oito núcleos físicos são recomendados para desempenho ideal
(c)	Os requisitos de memória são muito específicos para as necessidades de armazenamento do aplicativo e o número de nós no cluster. No entanto, 4 gigabytes deve ser considerada uma configuração mínima
(d)	O VoltDb suporta JDKs de OpenJDK ou Oracle/Sun.
(e)	NTP minimiza diferenças de tempo entre nós em um cluster de banco de dados, que é crítico para VoltDB. Todos os nós do cluster deve ser configurado para sincronizar contra o mesmo servidor NTP. Usando um

	único servidor NTP local é recomendado, mas não é obrigatório

3) Instalação

O VoltDB está disponível para download no site oficial. Ele possui uma versão open-source sob a licença GPL disponível gratuitamente, e uma versão paga anualmente. O site é https://www.voltdb.com/download

VoltDB é distribuído em um arquivo tar compactado. O nome do arquivo identifica a edição (comunidade ou empresa) e o número da versão. A melhor maneira de instalar VoltDB é descompactar o kit de distribuição como uma pasta no diretório home da conta pessoal, assim:
$ tar -zxvf voltdb-ent-6.5.tar.gz -C $HOME/

Instalando no diretório pessoal dá acesso total ao software e é mais útil para o desenvolvimento.

Se for instalar VoltDB em um servidor de produção onde o banco de dados será executado, pode-se instalar o software em um local padrão do sistema para que o cluster de banco de dados seja iniciado com os mesmos comandos em todos os nós. Os seguintes comandos shell instalam o software VoltDB na pasta /opt/voltdb:

```
$ sudo tar -zxvf voltdb-ent-6.5.tar.gz -C /opt
$ cd /opt
$ sudo mv voltdb-ent-6,5 voltdb
```

Nota: instalar como root usando o comando sudo faz com que as pastas de instalação fiquem somente como leitura para contas não privilegiadas. É por isso que a instalação em $ HOME é recomendada para executar os aplicativos de exemplos e outras atividades de desenvolvimento.

4) Instruções Básicas

4.1) Criação de Banco de Dados

Em VoltDB a definição do esquema de banco de dados é usando linguagem de definição de dados SQL declarações (DDL) como os outros bancos de dados SQL. Por exemplo, para criar uma tabela de cidades:

```
CREATE cidades table (   cidade VARCHAR (60),
                         uf VARCHAR (2)
                         );
```

O esquema anterior define uma única tabela com duas colunas: cidade e estado. Pode-se também incluir um conjunto de opções como valores padrão e chaves primárias.

4.2) Comandos CRUD

O VoltDb utiliza a sintaxe SQL em *store procedures* e em consultas *Ad Hoc*. A seguir serão apresentados alguns comandos. Maiores detalhes podem ser obtidos no documento UsingVoltDBdisponibilizado no site do VoltDB. http://www.voltdb.com

Instruções SQL de CRUD:
- Insert
- Select
- Update e Upsert
- Delete e Truncate

4.2.1) INSERT

Cria novas linhas no banco de dados, usando os valores especificados para as colunas.

Sintaxe

INSERT INTO *table-name* [(*column-name* [,...])] VALUES (*value-expression* [,...])
INSERT INTO *table-name* [(*column-name* [,...])] SELECT *select-expression*

Descrição

A instrução INSERT cria uma ou mais novas linhas no banco de dados. Existem duas formas da inserção:

INSERT INTO ... VALUES e INSERT INTO ... SELECT.

A instrução‘INSERT INTO ... VALUES’permite inserir valores específicos para a adição de uma única linha no banco de dados.

A instrução‘INSERT INTO ...SELECT’ permite inserir várias linhas no banco de dados, dependendo do número de linhas retornado pela expressão select.

4.2.2) SELECT

Obtém as linhas e colunas especificadas do banco de dados.

Sintaxe

Select-statement [{**set-operator**} ***Select-statement***] ...

Select-statement:
SELECT [TOP *valor inteiro*]
{ * | [ALL | DISTINCT] { *nome-da-coluna*| expressão-da-seleção } [AS *alias*] [,...] }
FROM { ***tabela-de-referência***} [**cláusula-do-*join***]...
[WHERE [NOT] expressão-boleana [{AND | OR} [NOT] expressão-boleana...]
[**cláusula**...]

tabela-de-referência:
{ nome-da-tabela[AS *alias*] | nome-da-view[AS *alias*] | ***sub-query***AS *alias*}
sub-query:
(***Select-statement***)

cláusula-do-*join*:
,***table-reference***
[INNER | {LEFT | RIGHT | FULL } [OUTER]] JOIN [{***table-reference***}] [***join-condição***]

join-condição:
ON *conditional-expression*
USING (*column-reference* [,...])

cláusula:
ORDER BY { *column-name* | *alias* } [ASC | DESC] [,...]
GROUP BY { *column-name* | *alias* } [,...]
HAVING *boolean-expression*
LIMIT *integer-value* [OFFSET *row-count*]

set-operator:
UNION [ALL]
INTERSECT [ALL]
EXCEPT

Descrição

A instrução SELECT recupera as linhas e colunas especificadas do banco de dados, filtradas e classificadaspor quaisquer cláusulas que estão incluídas no comando.

Na sua forma mais simples, a instrução SELECT recupera os valores associados com colunas individuais. No entanto, a expressão de selecção pode ser uma função talcomo COUNT e SUM.

As seguintes características e limitações são importantes para atentar quando usar a instrução SELECT comVoltDB:

• VoltDB suporta os seguintes operadores em expressões: adição (+), subtração (-), multiplicação (*),divisão (*) e de concatenação (||).

• TOP n é um sinônimo para LIMIT n.

• A expressão WHERE suporta os operadores booleanos: igual (=), não igual(= ou <>!), maior que (>), Menor do que (<), maior do que ou igual a (> =), inferior a ou igual a (<=), LIKE, IS NULL, IS DISTINCT,IS NOT DISTINCT, AND, OR e NOT. Nota: embora OR é suportado sintaticamente,O VoltDB não otimiza essa operação e uso de OR pode afetar o desempenho das consultas.

• Não são suportados resultados com grandes conjuntos de dados, maior que 50 megabytes de tamanho. Se for executada uma instrução SELECT que gera um conjunto de resultados de mais de 50 megabytes,o VoltDB retornará um erro.

4.2.3) UPDATE e UPSERT

UPDATE

A instrução UPDATE atualiza o valor de coluna(s) e linha(s) especificada(s).
UPDATE — Updates the values within the specified columns and rows of the database.

Sintaxe

UPDATE table-name SET column-name = value-expression [, ...] [WHERE [NOT] boolean-expression [{AND | OR} [NOT] boolean-expression]...]
Description

Descrição

A instrução UPDATE altera o valor de coluna(s) dentro de um grupo de registros especificados (linhas).
As seguintes imitações são importantes observar ao usar a instrução UPDATE com VoltDB:

• VoltDB suporta os seguintes operadores em expressões: adição (+), subtração (-), multiplicação (*), divisão (*).

• A expressão WHERE suporta os operadores booleanos: igual (=), não igual (!= ou <>), maior que (>), Menor do que (<), maior do que ou igual a (> =), inferior a ou igual a (<=), IS NULL, AND OR ou NOT.

Nota: embora OR é suportado sintaticamente, O VoltDB não otimiza essa operação e uso de OR pode afetar o desempenho das consultas.

UPSERT

A instrução UPSERT insere novas linhas ou atualiza linhas existentes dependendo do valor da chave primária.

Sintaxe
UPSERT INTO table-name [(column-name [,...])] VALUES (value-expression [,...])
UPSERT INTO table-name [(column-name [,...])] SELECT select-expression Description

Descrição
A instrução UPSERT tem a mesma sintaxe da instrução INSERT e executa da mesma forma, ou seja assume que um registro com a mesma chave primária não existe no banco de dados. Se o registro existir, UPSERT atualiza o registro existente com os novos valores da(s) coluna(s) informada(s).

Nota: A instrução UPSERT só pode ser executada em tabelas que possuem chave primária.

UPSERT tem as mesmas duas formas como a instrução INSERT: UPSERT INTO ... valores e UPSERT INTO ... SELECT.

A instrução UPSERT também tem restrições e limitações semelhantes como a instrução INSERT que diz respeito à associação de tabelas particionadas e cláusulas SELECT excessivamente complexas.

Entretanto, UPSERT INTO ... SELECT tem uma limitação adicional: a instrução SELECT deve produzir determinantemente resultados ordenados. Isto é, a consulta deve não só apresentar as mesmas linhas, elas devem estar na mesma ordem a fim de garantir as inserções e atualizações subsequentes e produzir resultados idênticos.

4.2.4) DELETE e TRUNCATE

DELETE

Exclui um ou mais registros do banco de dados

Sintaxe

DELETE FROM *table-name*
[WHERE [NOT] boolean-expression [{AND | OR} [NOT] boolean-expression]...]
[ORDER BY {*column-name* [ASC | DESC]}[,...] [LIMIT *integer*] [OFFSET *integer*]]

Descrição

A instrução *DELETE* exclui linhas da tabela especificada que atendam as restrições da cláusula *WHERE*.
As seguintes observações são importantes para quando usar a instrução *DELETE* em VoltDB:

• A instrução *DELETE* pode operar em apenas uma tabela de cada vez (sem *JOIN* ou subconsultas).

• A cláusula *WHERE* suporta os operadores booleanos: igual (=), não igual, (= ou <>!),maior que (>), Menor do que, (<), maior do que ou igual a (> =), inferior a ou igual a (<=), for nula, AND, OR e NOT.

Nota: embora OR é suportado sintaticamente, O VoltDB não otimiza essa operação e uso de OR pode afetar o desempenho das consultas.

• A cláusula ORDER BY permite ordenar os resultados de seleção e, em seguida, selecionar um subconjunto ordenado de registos a excluir. Por exemplo, pode-se excluir apenas os cinco registros mais antigos, cronologicamente, por timestamp:

DELETE FROM eventos ORDER BY event_time LIMIT ASC 5;

Da mesma forma, pode-se optar por manter apenas os cinco mais recente:

DELETE FROM eventos ORDER BY DESC event_time OFFSET 5;

TRUNCATE TABLE

A instrução TRUNCATE TABLE exclui todos os registros de uma tabela específica

Sintaxe

TRUNCATE TABLE nome-da-tabela

Descrição

A instrução TRUNCATE TABLE exclui todos os registros de uma tabela específica. Sua funcionalidade é a mesma da instrução DELETE FROM {nome-da-tabela} sem cláusula de seleção/condição. Esta instrução contém otimizações para melhorar a performance e reduzir o uso da memória se comparado a instrução DELETE com a cláusula WHERE.

As seguintes observções são importantes para se lembrar quando usar a instrução TRUNCATE TABLE em VoltDB:

Executar uma consulta TRUNCATE TABLE em uma tabela de partição dentro de uma *stored procedure single-partitioned* só irá excluir os registros dentro da partição atual. Registros em outras partições não serão afetados.
Não se pode executar a instrução TRUNCATE TABLE em uma tabela replicada com *stored procedure single-partitioned*. Para truncar uma tabela replicada deve-se executar com uma *store procedure multi-partition*ou como uma *query* ad hoc.

5) Tarefas administrativas

5.1) Backup e Restore

Há momentos em que é necessário guardar o conteúdo de um banco de dados VoltDB para o disco e, em seguida, restaurá-lo. Por exemplo, se o cluster precisa ser desligado para manutenção, é necessário salvar o estado atual do banco de dados antes de desligar o cluster e, em seguida, restaurar o banco de dados uma vez que o cluster volta a ficar online. Executar backups periódicos dos dados também pode fornecer um retorno em caso de falhas inesperadas, como por exemplo, falhas físicas ou até interrupções de energia.

VoltDB fornece comandos shell, procedimentos do sistema e um recurso instantâneo automatizado para ajudar executar essas operações.

Salvar e restaurar manualmente um banco de dados VoltDB é útil quando se precisa precisar fazer manutenção no próprio banco de dados ou no cluster onde está executando.

Para realizar os procedimentos de backup e restore deve seguir alguns procedimentos, tais como:

1. Pausar as atividades no banco de dados.
2. Salvar para escrever um "snapshot" para o disco com os dados atualizados.
3. Desligar o cluster.

4. Fazer alterações no esquema (“schema”) para o VoltDB, configurar o Cluster e/ou implementar arquivos conforme a necessidade.
5. Reiniciar o cluster no modo de administrador.
6. Recarregar o esquema(“schema”)e os procedimentos armazenados.
7. Restaurar o “snapshot” anterior.
8. Reiniciar a atividade do cliente.

É imprescindível que todas as atividades do banco sejam interrompidas antes do desligamento e do salvamento. Isso garante que nenhuma outra alteração no banco de dados será feita (e, portanto, perdida) após a salvar e antes do desligamento. Da mesma forma, é importante que nenhuma atividade cliente se inicie até que o banco de dados seja iniciado e a operação de restauração esteja concluída.

As operações de backup e restore são realizadas através do sistema do VoltDB ou usando os comandos shellVoltadmin. Na maioria dos casos, os comandos shell são mais simples, uma vez que não necessitam de código de programa para usar.

Quando é feito um comando de backup, deve-se especificar um caminho onde os dados serão salvos e um identificador exclusivo para marcar os arquivos.OVoltDB salva os dados atuais em cada nó do cluster para um conjunto de arquivos no local especificado (usando o identificador único como um prefixo para os nomes de arquivo). Este conjunto de arquivos é referido como um snapshot, uma vez que contém um registo completo da base de dados para um dado instante (quando a operação de gravação foi executada).

A opção “blocking” permite especificar se a operação de salvamento deve bloquear outras transações até que ela seja concluída.

Todos os nós do cluster usam o mesmo caminho absoluto, então o caminho especificado deve ser válido, deve existir em cada nó, e nenhum dado salvo anteriormente pode conter o mesmo identificador, caso isso ocorra a operação irá falhar.

Quando o comando de restauração é feito, deve ser especificado o mesmo caminho e o mesmo identificador único usado ao criar o snapshot. O VoltDB verifica para garantir que o set salvo existe em cada nó e em seguida restaura os dados na memória.

5.2) Segurança (autenticação, usuários, permissões)

A segurança é um aspecto importante de qualquer aplicação. Por padrão, VoltDB não executa qualquer verificação de segurança quando um aplicativo cliente abre uma conexão com o banco de dados ou chama uma *stored*

procedure. Isto é conveniente quando se desenvolve e distribui um aplicativo em uma rede privada.

No entanto, em redes públicas ou semi-privados, é importante para garantir que somente os aplicativos clientes conhecidos interajam com o banco de dados. VoltDB permite controlar o acesso ao banco de dados por meio das configurações nos arquivos de esquema e implantação. Quando um aplicativo cria uma conexão com um banco de dados VoltDB (usando ClientFactory.clientCreate), ele passa um nome de usuário e senha como parte da configuração do cliente. Estes parâmetros identificam o cliente para a base de dados e são utilizados para autenticar o acesso.

Em tempo de execução, se a segurança estiver ativada, o nome de usuário e senha passado pelo aplicativo cliente são validados pelo servidor contra os usuários definidos no arquivo de implantação. Se o aplicativo cliente passa o conjunto nome de usuário e senha válidos, a conexão é estabelecida. Quando o aplicativo chama um procedimento armazenado, as permissões são verificadas novamente. Se o esquema identifica o usuário como tendo permissão para ter acesso a*stored procedure*, o procedimento é executado. Se não, um erro é retornado para o aplicativo de chamada

Integrando Kerberos de segurança com VoltDB:

Para ambientes onde é necessária a comunicação mais segura que usernames com hash e senhas, é possível para um banco de dados VoltDB usar o Kerberos para autenticar clientes e servidores.
Kerberos é um popular protocolo de segurança de rede para autenticar os processos do cliente Java quando eles se conectam servidores de banco de dados VoltDB. Uso de Kerberos é suportado somente para a biblioteca cliente Java e uma interface JSON.

6) Exportaçao e Importação de Dados

O processamento de transações é muitas vezes apenas um dos aspectos no contexto maior de negócios e os dados precisam fazer a transição de um sistema para outro como parte da solução global. O processo de mover de um banco para outro e como os dados se movem através do sistema é muitas vezes referida como Extract, Transform, and Load (ETL). VoltDB suporta ETL através da capacidade de exportar dados de maneira seletiva, bem como a capacidade de importar dados através de múltiplos protocolos padrões.

A exportação de dados difere de salvar e restaurar de várias maneiras, a saber:
- São só exportados os dados selecionados (como exigido pelo processo de negócios);

- A exportação é um processo contínuo, em vez de um evento one-time.
- O resultado da exportação de dados é a informação que é utilizada por outros processos de negócio, e não como uma cópia de backup para restaurar o banco de dados;

Para importação, VoltDB suporta tanto a importação de uma só vez através de utilitários de carregamento de dados, bem como a importação em curso como parte do processo de banco de dados.

7) Principais Clientes

Nokia, Ericson, HP, Mitsubishi, Openet, Myfox, Emagine, MaxCdn, Sakura Internet Inc, Air Push, AsiaInfo, Novatel, Flytxt, PingerShopzilla, deltaDNA, SignMeUp, Bursa Malaysia, Peak Games

8) Conclusões

O VoltDB é um banco de dados relacional que vêm atingir um ramo onde os RDBMS estavam perdendo mercado para os NoSQL, que é o mercado de escalabilidade. Ele une os mundos do relacional com os de volume de dados,com as da necessidade de obtenção de dados de forma rápida e o da escalabilidade.

Através do particionamento de dados unificados com as instruções em *stored procedures* na mesma partição permite uma velocidade maior na obtenção das informações sem perder a consistência. O fato de utilizar as linguagens DDL e SQL permite uma mais rápida absorção pelos administradores de banco de dados e pelos desenvolvedores, até mesmo pelos usuários que já estão acostumados com a utilização de queries em SQL. Além disto, a escalabilidade é realizada através de aumento de servidores.

Por fim, vale lembrar de alguns pontos importantes para a boa utilização do VoltDB.

Melhores práticas:

1-Particionar as tabelas. Particionamento permite redimensionar dinamicamente o tamanho do seu banco de dados. Ele também permite o processamento simultâneo de transações.

2. Particionar as*stored procedures*. Para maximizar a produção, maximizar a frequência das operações partição única e minimizar multipartição das transações.

3. Conectar-se a todos os nós do cluster. A criação de múltiplas conexões de cliente evita possíveis gargalos e permite VoltDB para aproveitar a afinidade do cliente (no cliente Java).

4. Usar um número ímpar de nós. VoltDB protege contra a perda de dados devido a falha de rede ou do nó da replicação (chamado KSafety). Usando um número ímpar de nós no cluster maximiza a proteção e disponibilidade. Por exemplo, com dois nós de K = 1 cluster, caso se verifique uma falha de nó, há uma possibilidade de 50/50 do outro nó parar a fim de evitar uma parada na rede. Com três nós K = 1 cluster isso não acontecerá, a menos que dois nós falhem.

O que não se deve fazer:

1. Não retornar quantidades excessivas de dados de *Store Procedures*. Conjuntos de resultados muito grandes podem aumentar a latência da rede e impactar a performance. VoltDB limita os resultados das procedureas para 50MB. Manter os resultados em conjuntos pequenos para maximizar o rendimento e minimizar a latência.

2. Não criar tabelas enormes. VoltDB funciona melhor com pequenas a médias tabelas. Criação de tabelas com centenas de colunas ou muitas colunas grandes podem impactar o desempenho da consulta. É melhor criar várias tabelas menores. VoltDB limita colunas para 1MB cada uma e todas as colunas em uma única tabela de 2MB.

3. Não fazer processamento desnecessário em nas*stored procedures*ou rotinas de retorno de chamada. Manter as procedures otimizadas e bem definidas para evitar procedimentos de longa duração e bloquear outras transações. Além disso, rotinas de retorno de chamada são processadas uma de cada vez dentro do cliente, portanto manter os procedimentos de retorno de chamada eficientes para evitar parar o cliente após a conclusão da transação.

4. Não executar operações não determinísticas em *stored procedures*. VoltDB exige resultados determinísticos para garantir a consistência entre as partições e durante a recuperação. Evitar quaisquer operações em *stored procedures que*dão resultados variáveis, como a hora do sistema, números aleatórios, arquivo de I / O, etc. Em vez disso, gerar os dados variáveis no cliente e usá-lo como entrada para o procedimento.

9) Referências Bibliográficas

http://db-engines.com/en/ranking
https://451research.com/state-of-the-database-landscape
http://www.voltdb.com
https://docs.voltdb.com/UsingVoltDB
Edward Ribeiro - https://www.infoq.com/br/presentations/voltb-arquitetura-desenv