OrientDB

Greta Augat Abib[1]

[1]Universidade Federal do ABC (UFABC)

Resumo. *O OrientDB é uma solução de banco de dados NoSQL de código aberto, que permite a utilização de banco de dados orientados a grafos. Trata-se de uma solução versátil, pois inclui funcionalidades de banco de dados relacionais, mecanismos orientados a objetos, bancos de dados orientados a documentos e, modelos de grafos, sendo um banco de dados escalável e de alto desempenho.*

1. Introdução

Ao abordar banco de dados, o modelo mais comumente utilizado são as bases de dados relacionais. Uma base relacional possui a estrutura de tabelas, colunas, tuplas (linhas), chaves e relacionamentos. Para realizar o relacionamento entre uma tabela e outra, uma das tabelas possui a indicação de chave estrangeira (*foreign key*). E quando é necessário realizar um relacionamentos de muitos para muitos (*many to many)* é necessária uma terceira tabela ligando às outras duas para registrar seus relacionamentos.

Em um banco de dados orientado a grafos, relacionamentos são modelados de maneira mais intuitiva. Existem as entidades chamadas de vértices (ou nodes) que são ligadas entre elas pelas arestas (ou *edges*) cada um podendo guardar dados entre os relacionamentos e cada relacionamento pode ter uma direção.

Sistemas de banco de dados em grafos suportam aplicações baseadas em modelos de dados cuja a interconectividade de dados é um aspecto importante. O volume de dados crescente envolvendo tais aplicações vem sendo tratado por uma série de soluções de gerenciamento de dados baseados em grafos visando a escalabilidade destes sistemas. Além de prover uma forma direta para a representação de dados complexos, bancos de dados em grafos classificados como nativos sao capazes de executar consultas de forma eficiente por eliminar operações custosas de junções [Penteado et al. 2014].

Um exemplo clássico do uso de banco de dados em grafos é o Twitter [Twi], rede social que oferece serviço de microblog. Sua base é modelada como grafo, onde os usuarios são os vértices e os relacionamentos entre eles são as arestas. Outros exemplos sao: sistemas que recomendam compras em lojas virtuais, sistemas que exploram dados químicos e biológicos para detecção de padrões, e sistemas web como o PageRank [Page et al. 1998] da Google que analisa a importancia dos sites computando o numero de arestas incidentes em cada site.

Os sistemas de banco de dados em grafos modelam seus dados por meio de vértices e arestas facilitando a modelagem de contextos complexos e definindo naturalmente relações existentes entre as entidades de uma base. Nesta categoria os sistemas podem ser classificados como nativos ou não-nativos. Os nativos consideram a estrutura de grafo tanto no armazenamento físico dos dados quanto no processamento de consultas. Listas de adjacencias vértice-vértice são usadas no armazenamento físico possibilitando a exploração do grafo de forma simples, direta e eficiente no processamento de consultas.

Em contrapartida, os sistemas não-nativos usam outros modelos para o armazenamento e processamento de consultas. Por exemplo, por meio do modelo relacional, as relações de triplas vértice-aresta-vértice em um grafo são armazenadas como tuplas em tabelas. Este tipo de composicão é prejudicial para o desempenho de consultas quando diversas junções são necessárias para executar uma consulta complexa envolvendo diversas triplas [Penteado et al. 2014].

O OrientDB [Ori] é um banco de dados em grafo aberto implementado em Java, é transacional e dá suporte a arquitetura centralizada e distribuída com replicação. O OrientDB trabalha com banco de dados baseados em grafos nativo, o que reduz a latência nas operações de criação, leitura, atualização e exclusão (CRUD).

2. Arquitetura

2.1. Tipos de Dados

O principal recurso do OrientDB é suportar objetos de vários modelos como *Document*, *Graph*, *Key/Value* e *Real Object*. Ele contém uma API (*Application Program Interface*) separada para suportar todos esses quatro modelos.

2.1.1. Modelo de Documento *(Document Model)*

A terminologia *Document Model* pertence ao banco de dados NoSQL. Isso significa que os dados são armazenados nos documentos e o grupo de documentos é chamado de coleção. Tecnicamente, documento significa um conjunto de pares chave / valor ou também referidos como campos ou propriedades.

O OrientDB usa conceitos como classes, clusters e link para armazenar, agrupar e analisar os documentos.

A Tabela 1 ilustra a comparação entre modelo relacional, modelo de documento e o modelo de documento do OrientDB.

Relational Model	**Document Model**	**OrientDB Document Model**
Table	Collection	Class or Cluster
Row	Document	Document
Column	Key/value pair	Document field
Relationship	Not available	Link

Tabela 1

2.1.2. Modelo Baseado em Grafos *(Graph Model)*

Uma estrutura de dados baseada em grafos é um modelo de dados que pode armazenar dados na forma de vértices *(vertex)* interconectados por arestas *(edges)*. A ideia do banco de dados orientados a grafos do OrientDB veio do grafo de propriedades. O vértice e a aresta são os principais artefatos do modelo baseado em grafos. Eles contêm as propriedades que se assemelham ao modelo de documentos.

A Tabela 2 mostra a comparação entre o modelo de dados relacional, o modelo de grafos e o modelo de grafos do OrientDB.

Relational Model	**Graph Model**	**OrientDB Graph Model**
Table	Vertex and Edge Class	Class that extends ”V”(for Vertex) and ”E”(for Edges)
Row	Vertex	Vertex
Column	Vertex and Edge property	Vertex and Edge property
Relationship	Edge	Edge

Tabela 2

2.1.3. O Modelo Chave/Valor *(Key/Value Model)*

O modelo *Key/Value* permite que os dados sejam armazenados na forma de par chave/valor, em que os valores podem ser de tipos simples e complexos. Pode suportar documentos e elementos de grafos como valores.

A Tabela 3 ilustra a comparação entre modelo relacional, modelo de Key/Value e modelo de Key/Value do OrientDB.

Relational Model	**Key/Value Model**	**OrientDB Key/Value Model**
Table	Bucket	Class or Cluster
Row	Key/Value pair	Document
Column	Not available	Document field or Vertex/Edge property
Relationship	Not available	Link

Tabela 3

2.1.4. O Modelo de Objeto *(Object Model)*

Este modelo foi herdado pela programação Orientada a Objetos e suporta herança entre tipos (subtipos estendem os super-tipos), Polimorfismo quando se refere a uma classe base e Direct binding de/para *from/to* objetos usados em linguagens de programação.

A Tabela 4 ilustra a comparação entre modelo relacional, modelo de objeto e modelo de objeto do OrientDB.

Relational Model	**Object Model**	**OrientDB Object Model**
Table	Class	Class or Cluster
Row	Object	Document or Vertex
Column	Object property	Document field or Vertex/Edge property
Relationship	Pointer	Link

Tabela 4

2.2. Terminologia do OrientDB

A seguir são descritas algumas das terminologias utilizadas pelo OrientDB:

2.2.1. Registro (*Record*)

A menor unidade que se pode carregar e armazenar no banco de dados. Os registros podem ser armazenados em quatro tipos.

- *Document*
- *Record Bytes*
- *Vertex*
- *Edge*

Quando o OrientDB gera um registro, o servidor de banco de dados atribui automaticamente um identificador de unidade ao registro, chamado RecordID (RID). O RID possui a seguinte forma: $< cluster >:< position >$, sendo que $< cluster >$ significa o número de identificação do cluster e $< position >$ significa a posição absoluta do registro no cluster.

2.2.2. Documento (*Document*)

O *document* é o tipo de registro mais flexível disponível no OrientDB. Os documentos são digitados suavemente e são definidos por classes de esquema com restrição definida, mas também é possível inserir o documento sem nenhum esquema, ou seja, ele também suporta o modo sem esquema.

Os documentos podem ser facilmente manipulados por exportação e importação no formato JSON. O exemplo JSON 1 define as propriedades do documento *(document details)*:

Listing 1. Detalhes do formato Documento - JSON

```
{
    "id"          : "1201",
    "name"        : "Jay",
    "job"         : "Developer",
    "creations" : [
        {
            "name"    : "Amiga",
            "company" : "Commodore Inc."
        },

        {
            "name"    : "Amiga 500",
            "company" : "Commodore Inc."
        }
    ]
}
```

2.2.3. *RecordBytes*

O tipo de registro RecordBytes é o mesmo que o tipo BLOB no banco de dados relacional. O OrientDB pode carregar e armazenar o tipo de registro do documento junto com dados binários.

2.2.4. Vértice *(Vertex)*

O banco de dados do OrientDB não é apenas um banco de dados de documentos, mas também um banco de dados baseado em grafos. Os novos conceitos, como Vertex e Edge, são usados para armazenar os dados na forma de grafos. Em bancos de dados orientados a grafos, a unidade de dados mais básica é o nó, que no OrientDB é chamado de vertex. O Vertex armazena informações para o banco de dados.

2.2.5. Aresta *(Edge)*

Existe um tipo de registro separado chamado Edge que conecta um vértice a outro. As arestas são bidirecionais e só podem conectar dois vértices. Existem dois tipos de arestas no OrientDB, uma é regular e outra é leve.

2.2.6. Classe *(Class)*

A classe é um tipo de modelo de dados e o conceito extraído do paradigma de programação orientada a objetos. Com base no modelo de banco de dados de documentos tradicional, os dados são armazenados na forma de coleção, enquanto os dados do modelo de banco de dados relacional são armazenados em tabelas. O OrientDB segue a Document API junto com o paradigma OPPS. Como conceito, a classe no OrientDB tem o relacionamento mais próximo com a tabela nos bancos de dados relacionais, mas (ao contrário das tabelas) as classes podem ser sem esquema, repletas de esquema ou mistas. As classes podem herdar de outras classes, criando árvores de classes. Cada classe tem seu próprio cluster ou clusters (criados por padrão, se nenhum for definido).

2.2.7. *Cluster*

Cluster é um conceito importante que é usado para armazenar registros, documentos ou vértices. Em palavras simples, o Cluster é um local onde um grupo de registros é armazenado. Por padrão, o OrientDB criará um cluster por classe. Todos os registros de uma classe são armazenados no mesmo cluster com o mesmo nome da classe. Você pode criar até $32.767 (2^{15-1})$ clusters em um banco de dados.

A classe CREATE é um comando usado para criar um cluster com nome específico. Depois que o cluster é criado, é possível utilizar o cluster para salvar registros especificando o nome durante a criação de qualquer modelo de dados.

2.2.8. Relacionamentos *(Relationship)*

O OrientDB suporta dois tipos de relacionamentos: referenciados e incorporados. Relacionamentos referenciados significa que ele armazena o link direto para os objetos de destino dos relacionamentos. Nos relacionamentos incorporados, o relacionamento é armazenado dentro do registro que o incorpora. Esse relacionamento é mais forte que o relacionamento de referência.

3. Problemas

3.0.1. Arquitetura Distribuida

O OrientDB pode ser distribuído em diferentes servidores e usado de maneiras diferentes para obter o máximo de desempenho, escalabilidade e robustez.

O OrientDB usa o projeto *Hazelcast Open Source*[haz] para a descoberta automática de nós, armazenando a configuração de cluster de tempo de execução e sincronizando determinadas operações entre nós.

Em 2012, o OrientDB utilizava o modelo de Master/Slave para replicação dos dados. Embora esse modelo escalasse (*scale up*) bem nas leituras, os usuários reclamavam de gargalo (*bottleneck*) no nó mestre conforme demonstrado na Figura 1. É relativamente fácil escalar para as leituras, a parte difícil é escalar tanto para leituras como gravações.

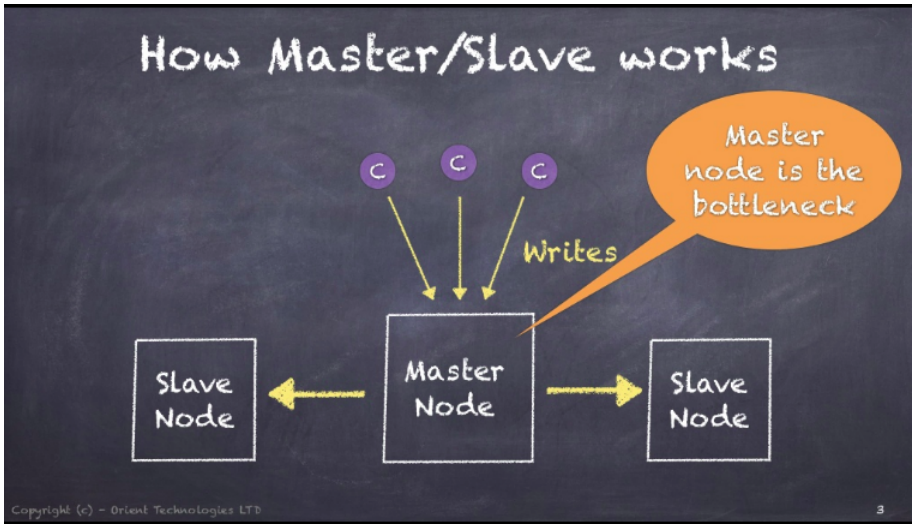

Figura 1. Master-Slave

Prós e contras do modelo *Master/Slave* do OrientDB:

Prós

- Relatividade fácil de desenvolver

Contras:

- O mestre é o gargalo para as gravações (*writes*).
- Não importa a quantidade de servidores, a taxa de transferência (*throughput*) é limitada pelo nó mestre.

Devido aos problemas identificados na arquitetura *Master/Slave* do OrientDB, a partir de 2012 foi implementada uma nova arquitetura distribuída com os seguintes objetivos:

- **Multi-mestre (*Multi-Master*):** todos os nós devem aceitar escritas
- Sharding: divide dados em várias partições
- Melhor failover
- Configuração simplificada com detecção automática (*Auto-Discovery*)

A Figura 2 apresenta o modelo de Master-Node com Auto-Discovery.

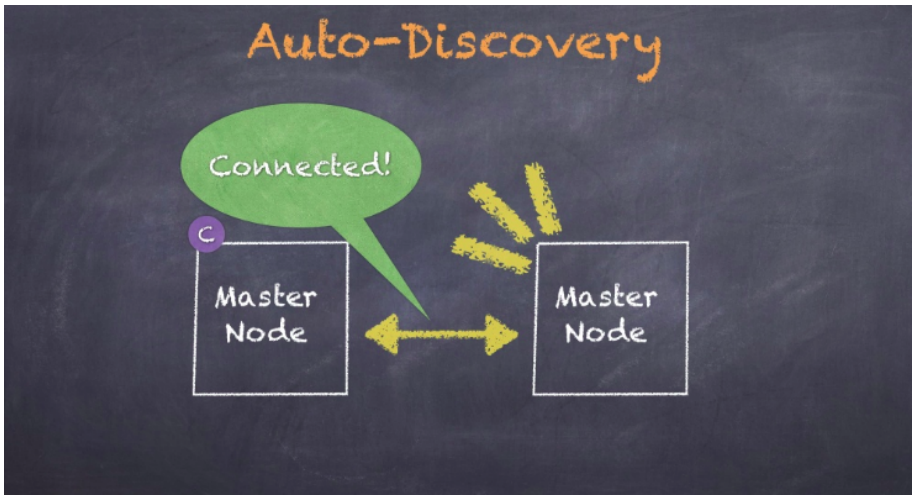

Figura 2. Master-Node

A Figura 3 exemplifica como a configuração é distribuída entre todos os clientes na arquitetura distribuída utilizando Multi-mestre.

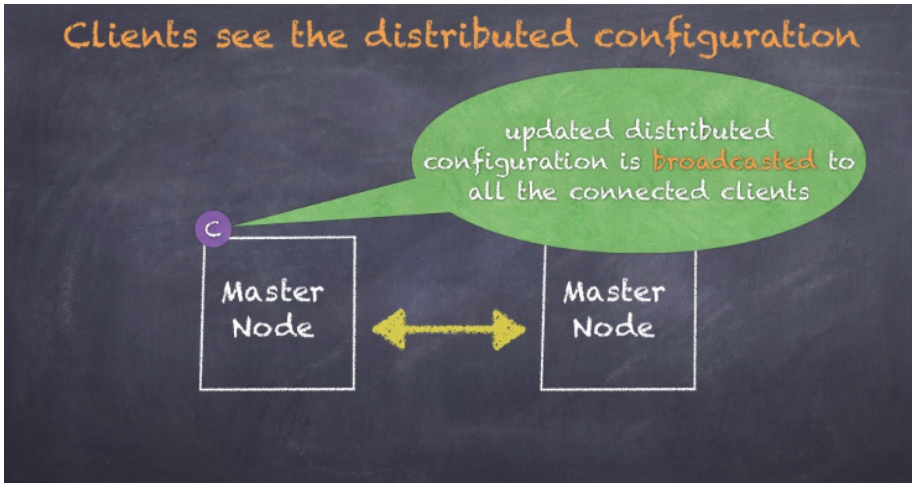

Figura 3. Configuração distribuída

O OrientDB utiliza uma solução semelhante ao **RAID** (Redundant Array of Independent Disks) para HardDrives.

Regras utilizadas provenientes do Hazelcast *Hazelcast Open Source*[haz]:

- Auto descoberta - *Auto discovering* (Multicast / TCP-IP / Amazon)
- Filas para solicitações e respostas
- Armazenar metadados em mapas distribuídos
- Bloqueios Distribuídos (*Distributed Locks*)

3.0.2. Criação de registros na Arquitetura Distribuída

Criação de registros (documentos, vértices e arestas) No modo distribuído, o RID é atribuído com a localidade do cluster. Se você tiver classe Cliente e 3 nós (node1, node2, node3), você terá estes clusters:

```
cliente com id = # 15 (padrao, atribuido ao node1)
customer_node2 com id = # 16
customer_node3 com id = # 17
```

Portanto, ao criar um novo cliente no node1, ele obterá o RID com cluster-id de cluster $"cliente" : 15$. A mesma operação no node2 gerará um RID com cluster-id = 16 e 17 no node3.

Dessa forma, o RID nunca colide e cada nó pode ser um mestre na inserção sem nenhum conflito.

3.0.3. Transações Distribuídas

O OrientDB suporta transações distribuídas. Quando uma transação é confirmada, todos os registros atualizados são enviados em todos os servidores, portanto, cada servidor é responsável por confirmar a transação. No caso de um ou mais nós falharem na confirmação, o quorum será verificado. Se o quorum tiver sido respeitado, os nós com falha serão alinhados aos nós do vencedor, caso contrário, todos os nós reverterão a transação.

Ao executar de maneira distribuída, as transações usam um protocolo de bloqueio de duas fases, então não há bloqueios entre o início e o commit, mas tudo é executado apenas no tempo de commit.

Durante o tempo de commit, o OrientDB adquire bloqueios nos registros tocados e verifica a versão dos registros (abordagem MVCC otimista). Neste ponto isso pode acontecer:

- Todo o registro pode ser bloqueado e ninguém tocou nos registros desde o início da transação, então a transação sofre commit.
- Caso alguém modificou algum dos registros que fazem parte da transação, a transação falha e o cliente pode tentar novamente
- Se no momento da confirmação, outra transação bloqueia qualquer um dos mesmos registros, a transação falha, mas a nova tentativa nesse caso é automática e configurável.

Se você tem 5 servidores, e writeQuorum é a maioria $(N/2 + 1 = 3)$, as seguintes ações podem ocorrer:

- Todos os 5 servidores realizam o commit da transação

- 1 ou 2 servidores reportam qualquer erro, a transação ainda é comitada (o quorum passa) e 1 ou 2 servidores serão forçados a ter o mesmo resultado que os outros.
- 3 servidores ou mais têm diferente resultados/erros, portanto, a transação é revertido (*rollback*) em todos os servidores para o estado inicial.

Visibilidade durante a transação distribuída: durante a transação distribuída, em caso de reversão, pode haver um período de tempo em que os registros aparecem alterados antes de serem revertidos.

Server Roles

O OrientDB possui uma arquitetura distribuída multi-master (chamada também de ”master-less”) onde cada servidor pode ler e gravar. A partir da v2.1, o OrientDB suporta o papel de ”REPLICA”, onde o servidor está em modo somente leitura, aceitando apenas comandos idempotentes, como Reads e Query. Além disso, quando o servidor ingressa no cluster distribuído como ”REPLICA”, os próprios clusters de registro não são criados como os nós ”MASTER”.

A partir da v2.2, a maior vantagem de ter muitos servidores REPLICA é que eles não concorrem em writeQuorum, portanto, se você tiver 3 servidores MASTER e 100 servidores REPLICA, cada operação de gravação será replicada em 103 servidores, mas a maioria deles o writeQuorum seria apenas 2, porque, dado N / 2 + 1, N é o número de servidores MASTER. Neste caso, depois que a operação é executada localmente, o coordenador do servidor da operação de gravação deve aguardar apenas mais um servidor MASTER.

Política de Resolução de Conflitos

No caso de um número par de servidores ou quando o banco de dados não estiver alinhado, o OrientDB usa uma cadeia de Estratégia de Resolução de Conflitos. Esta cadeia padrão é definida como uma configuração global (distributed.conflictResolverRepairerChain):

-Ddistributed.conflictResolverRepairerChain=majority,content,version)

A implementação da Estratégia de Resolução de Conflitos é chamada em cadeia após a ordem da declaração até que um vencedor seja selecionado. Na configuração padrão (acima):

primeiro é verificado se existe uma maioria estrita para o registro em termos de versões de registro. Se a maioria existe, o vencedor é selecionado se nenhuma maioria estrita foi encontrada, o conteúdo do registro é analisado. Se a maioria for atingida através da criação de um registro com versões diferentes, mas com conteúdo igual, esse registro será o vencedor usando a versão superior entre eles. se nenhuma maioria foi encontrada com o conteúdo, então a versão superior ganha (supondo que uma versão superior signifique o maior número de atualizações) O OrientDB Enterprise Edition suporta a resolução adicional de conflito do centro de dados (DC).

No final da cadeia, se nenhum vencedor for encontrado, os registros não serão tocados e somente uma intervenção manual poderá decidir quem é o vencedor. Nesse caso, uma mensagem de ADVERTÊNCIA é exibida no console com o texto O reparo automático não pode encontrar um vencedor para o registro $< rid >$ e os seguintes grupos de conteúdo: $[< registros >]$.

Limitações do modo distribuído

O OrientDB v2.2.x possui algumas limitações quando utilizado no modo distribuído:

- Banco de dados *in-memory* não é suportado.
- A importação de um banco de dados durante a execução distribuída não é suportada. Importe o banco de dados no modo não distribuído e, em seguida, execute o OrientDB no modo distribuído.
- Em releases anteriores ao v2.2.6, a criação de um banco de dados em vários nós pode causar problemas de sincronização quando os clusters são criados automaticamente. Por favor, crie os bancos de dados antes de executar no modo distribuído.
- *Constraints* com bancos de dados distribuídos podem causar problemas porque algumas operações são executadas em 2 etapas: *create* + *update*. Por exemplo, em algumas circunstâncias, as arestas (*edges*) podem ser criadas primeiro e, em seguida, atualizadas, mas as restrições como MANDATORY e NOTNULL contra campos falharão na primeira etapa, tornando impossível a criação de arestas (*edges*) no modo distribuído.
- O Auto-Sharding não é suportado no significado comum da Tabela de Hash Distribuído (DHT). A seleção do fragmento certo (cluster) depende da aplicação. Esse ponto será tratado nas próximas versões.
- Índices Sharded ainda não são suportados, portanto, criar um índice UNIQUE contra uma classe sharded não garante que uma chave seja exclusiva. Isso será resolvido com o particionamento automático nas versões posteriores.
- A mudança a quente da configuração distribuída está disponível apenas na *Enterprise Edition* (licença comercial).
- Não completar o *merging* de resultados para todas as projeções quando executado na configuração de sharder. Algumas funções, como o AVG (), não funcionam no *map/reduce.*

4. Referências

Referências

Hazelcast. `https://hazelcast.com/`. Acessado em: 24/04/2019.

Orientdb. `https://orientdb.com/`. Acessado em: 20/04/2019.

Twitter. `https://twitter.com/`. Acessado em: 20/04/2019.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web.

Penteado, R. R., Schroeder, R., Hoss, D., Nande, J., Maeda, R. M., Couto, W. O., and Hara, C. S. (2014). Um estudo sobre bancos de dados em grafos nativos. *X ERBD-Escola Regional de Banco de Dados*.