ISCTE-IUL
Mestrado em Ciência de Dados

Relatório Teórico acerca de componentes do stack Hadoop

Trabalho de Grupo realizado no âmbito da Unidade Curricular de Bases de Dados Distribuídas Avançadas do 1º ano do Mestrado em Ciência de Dados

Diogo Freitas, 104841, MCD-LCD
daafs@iscte-iul.pt

João Francisco Botas, 104782, MCD-LCD
Joao_Botas@iscte-iul.pt

Rebeca Sampaio, 126628, MCD-LCD
rhmso1@iscte-iul.pt

6 de janeiro 2025
Versão 1.0.0

Índice

1. HDFS

HDFS (Hadoop Distributed File System) é um sistema de ficheiros distribuído projetado para armazenar grandes volumes de dados em *hardware* comum (*commodity hardware*). É constituído por blocos que são distribuídos e replicados, através de um ou mais nós de um *cluster*. O sistema segue o princípio WORM (Write Once, Read Many), ou seja, os dados armazenados são imutáveis e não podem ser atualizados após serem escritos.

Num **ambiente de produção** será feito o *upload* dos dados e os ficheiros serão divididos em blocos com uma determinada dimensão, geralmente de 128MB (conhecida também como a fase de *data ingestion*). Terá um NameNode que guarda os metadados, e onde depois a *client application* obtém deste NameNode a lista dos DataNodes. Os DataNodes armazenam os blocos de dados e fazem operações de leitura e escrita diretamente para o *client*. Pode ainda ser configurado um Secondary NameNode ou Standby NameNode para garantir alta disponibilidade e que esse outro NameNode assuma automaticamente em caso de falha principal de dados. Para garantir tolerância a falhas geralmente é definido um fator de replicação de 3[1].

O sistema distribuído em nós garante que grandes volumes de dados sejam armazenados e processados de maneira eficiente e aproveita os conceitos fundamentais de bases de dados distribuídas, como particionamento, na divisão de ficheiros em blocos de dados de tamanho fixo; replicação, para garantir a tolerância a falhas e alta disponibilidade; e escalabilidade, no sentido em que é escalável (vertical e horizontalmente) com base no tamanho do sistema do ficheiro. O HDFS, por estas razões, aborda o teorema CAP, um *trade-off* entre consistência, disponibilidade e tolerância a partições; no sentido em que se houver desconexão temporária entre os nós, os dados continuam acessíveis, inclusive os sistemas que dependem do HDFS.

2. YARN

O YARN[2] (Yet Another Resource Negotiator) é uma parte do Hadoop que foi criada para resolver problemas de desempenho na versão 1.0 do Hadoop, onde o **Job Tracker** ficava sobrecarregado[3]. Introduzido no Hadoop 2.0, o YARN é agora um sistema eficiente para processar grandes volumes de dados, conhecido como um "*Redesigned Resource Manager*".

O principal diferencial do YARN é que ele separa a gestão de recursos (quem usa o quê) do processamento de dados. No Hadoop 1.0, o **Job Tracker** fazia tudo sozinho, mas no YARN essa tarefa é dividida entre dois componentes:

- **Resource Manager** (RM): É o principal responsável por gerenciar os recursos do cluster. O RM decide como alocar recursos como memória, CPU e outros para as aplicações em execução. Além disso, o RM mantém o controle sobre os recursos disponíveis e assegura que as aplicações recebam os recursos necessários para a execução eficiente.
- **Application Manager** (AM): Cada aplicação submetida ao YARN possui um Application Manager dedicado. O AM gerencia a execução da aplicação, monitorando o estado e a execução das tarefas, garantindo que sejam processadas corretamente.

O **YARN** permite que diferentes motores de processamento de dados, como **grafos**, **processamento interativo** e em **lote**, processem dados no **HDFS** de forma eficiente. O **YARN** é **escalável**, oferecendo capacidade para gerenciar milhares de nós e clusters, e garante **compatibilidade** com aplicações MapReduce (Secção 3) existentes. O **YARN** também proporciona **otimização da utilização do cluster** através da alocação dinâmica de recursos e suporta **multi-tenancy**, permitindo que diferentes motores (diferentes *frameworks* ou sistemas de processamento) acessem e utilizem os mesmos recursos simultaneamente.

Numa **configuração em produção**, o **YARN** possui as seguintes características:

1. **ResourceManager** e **Application Manager** → referidos anteriormente

[1] no modo pseudo-distribuído, o fator é 1
[2] https://www.linkedin.com/pulse/hadoop-123-rui-cunha/
[3] https://jayvardhan-reddy-v.medium.com/bigdata-part3-hadoop-1-0-architecture-763f51a0f5f

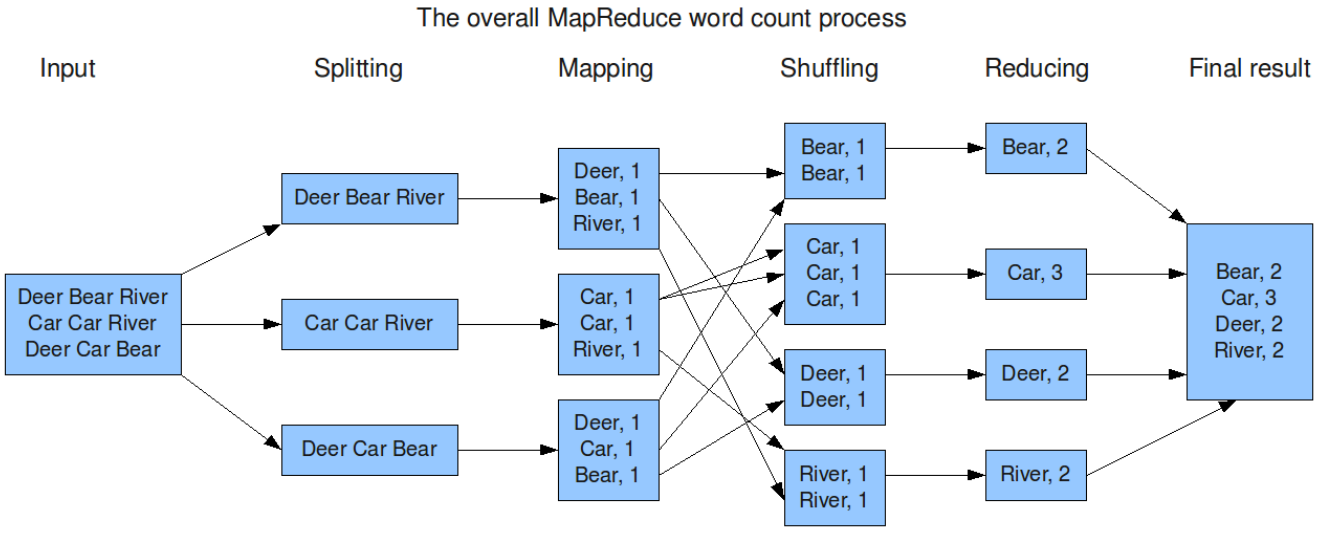
2. **NodeManagers**: O **Node Manager** gere os recursos e o funcionamento de cada nó (máquina) num cluster Hadoop.
   - Envia sinais periódicos sobre o estado do nó ao **Resource Manager**
   - Monitora o uso de recursos, gere logs e pode terminar containers conforme as instruções do **Resource Manager**.
   - Cria e inicia containers quando solicitado pelo **Application Master**
3. **ApplicationMaster**: Uma **aplicação** é um trabalho enviado para um sistema. O **Application Master** é responsável por pedir recursos ao **Resource Manager**, acompanhar o progresso e o estado da aplicação.
   - Solicita o container ao **Node Manager**, enviando tudo o que a aplicação precisa para funcionar;
   - Durante a execução, o **Application Master** envia relatórios periódicos sobre o estado da aplicação para o **Resource Manager**.
4. **Container**: Um **container** é um conjunto de recursos físicos, como memória RAM, núcleos de CPU e espaço em disco, em um único nó.

3. MapReduce

Modelo de programação para processamento paralelo de grandes volumes de dados. Inicializa com *input files* guardados no HDFS e separa em *splits*. Cada *split* é executado e depois separado nas fases: **Map** (+ Shuffle and Sort) e **Reduce**.

Num **ambiente de produção** executa tarefas em etapas de Map e Reduce, distribuídas nos nós do *cluster*.

1. ***Fase Map***: Dividir os dados de entrada em pequenas partes (*splits*) para serem processadas em paralelo nos nós do *cluster*. Produz pares chave-valor intermédios.
2. ***Fase Reduce***: Consolida os dados agrupados e ordenados, para aplicar funções de agregação.As tarefas *Reduce* são executadas em **containers** alocados pelo **YARN** nos nós do *cluster*. Os resultados finais são escritos no HDFS.



No MapReduce existe particionamento para dividir dados em blocos processados em paralelo na fase "Map" e redistribui por chaves para a fase de "Reduce". O MapReduce é essencial para dados distribuídos em larga escala (HDFS).

4. HBase

O HBase é uma solução de armazenamento e processamento de dados NoSQL. Diferente das bases de dados relacionais tradicionais, o HBase utiliza tabelas numa coleção de linhas organizadas por famílias de colunas, o que permite um desempenho otimizado ao lidar com conjuntos de dados de larga escala e consultas em colunas específicas. O HBase funciona sobre o Hadoop Distributed File System (HDFS) e aproveita a sua arquitetura distribuída para garantir alta disponibilidade e escalabilidade.

O HBase depende do Hadoop para fornecer um sistema de ficheiros distribuído, confiável e escalável. Portanto, antes de iniciar o HBase, é necessário configurar a infraestrutura, que inclui um *cluster* Hadoop com HDFS para armazenamento distribuído; o HBase Master para coordenar os *Region Servers*, responsáveis por armazenar e gerir tabelas e partições de dados, bem como processar operações de leitura e escrita. O **ZooKeeper** também desempenha um papel essencial para coordenar e distribuir a carga de trabalho nos nós do HBase (o que garante consistência). O HBase deve ser instalado em cada nó do *cluster*, com variáveis de ambiente como HBASE_HOME e *JAVA_HOME* devidamente configuradas. O ficheiro `hbase-site.xml` deve ser ajustado para definir o diretório do HDFS, otimizar limites de memória para *Region Servers* e habilitar alta disponibilidade (HA) para o HMaster. No `hbase-env.sh`, é importante ajustar os parâmetros de memória JVM para os serviços HBase, a fim de otimizar o desempenho.

O HBase utiliza escalabilidade para dividir tabelas em várias regiões, fragmentadas e distribuídas entre os nós do cluster. A replicação é gerida pelo HDFS, enquanto a fragmentação permite paralelismo e acesso eficiente aos dados nas diferentes regiões. Oferece consistência forte dentro de uma região e consistência eventual em operações globais. A alta disponibilidade é garantida pela realocação automática de regiões em caso de falhas, com logs de escrita a proteger as transações efetuadas.

5. Phoenix

O Phoenix é uma camada de abstração SQL desenvolvida para funcionar diretamente sobre o HBase. Combina a flexibilidade e escalabilidade do HBase com a familiaridade e simplicidade das consultas SQL (mapeia tabelas HBase) e faz transações **ACID**[4].

Para o ambiente de produção, é fundamental entender os componentes do Phoenix que garantem a integração com o HBase. Como componentes existem: o Cliente Phoenix, por meio do ***driver* JDBC**, facilita conexões entre os dados do Phoenix e aplicações que suportem conectividade JDBC; o Phoenix Query Server permite a execução escalável das consultas em ambientes distribuídos; o Phoenix Compiler otimiza as consultas SQL, traduzindo-as em comandos eficientes para o HBase. Associado à replicação, o Phoenix aproveita o suporte nativo do HBase para replicação de dados, ao permitir consultas SQL diretamente em réplicas secundárias. Isso não apenas aumenta a disponibilidade de dados, mas também melhora significativamente o desempenho de leitura em sistemas de produção.

Do HBase, o Phoenix herda a escabilidade e o processamento paralelo, ou seja, organiza as tabelas em regiões distribuídas por vários nós do *cluster*. A replicação, gerida pelo HDFS subjacente, garante alta disponibilidade e tolerância a falhas, enquanto *logs* de escrita asseguram a recuperação de transações; as transações ACID asseguram a integridade e confiabilidade dos dados.

[4] Atomicidade, Consistência, Isolamento, Durabilidade

6. Hive

O **Hive** é um sistema de armazenamento e consulta de dados para o ecossistema **Apache Hadoop**, desenvolvido inicialmente pelo **Facebook**. É usado principalmente como uma solução de **data warehouse** em ambientes Hadoop, permitindo a análise e consulta de grandes conjuntos de dados armazenados no **HDFS**. O Hive utiliza uma linguagem de consulta chamada **HiveQL**, que é semelhante ao **SQL**, facilitando a interação com os dados para analistas que possuem conhecimento em SQL, mas não necessariamente em programação Java.

O Hive implementa uma abstração sobre os dados no HDFS, permitindo que sejam acedidos de forma mais simples através de comandos **DML** (Data Manipulation Language), como em bases de dados tradicionais. Ele também conta com um componente importante chamado **Metastore**, que armazena metadados sobre a estrutura e localização dos dados, ajudando na organização e otimização das consultas. Apesar de ser uma solução poderosa, o Hive apresenta algumas limitações em comparação com sistemas de bases de dados tradicionais. Por exemplo:

- **UPDATE** não é suportado.
- Não existem transações, *rollbacks* ou níveis de isolamento transacional.
- Não há suporte para **chaves primárias**, **estrangeiras** ou outras restrições de integridade declarativas.
- Dados mal formatados são representados como **NULL**.

"Desde a versão 0.14, o Hive oferece suporte a **transações ACID**, utilizando **MapReduce** ou **Tez** para garantir confiabilidade no processamento de dados."

O Hive é composto por vários componentes, incluindo:

- **Metastore**: armazena os metadados das tabelas.
- **Driver**: responsável pela compilação e otimização das consultas.
- **Engine**: executa as consultas, usando **MapReduce** ou **Spark**.

No que diz respeito à **fragmentação**, o Hive suporta o **particionamento** de tabelas para melhorar a organização dos dados e também permite o ***bucketing***, que ajuda na melhor distribuição dos dados. Além disso, o Hive otimiza as consultas através de ***pruning* de partições**, ou seja, só processando as partições relevantes para a consulta.

7. Hue

O Hue (Hadoop User Experience) é uma interface gráfica de utilizador de código aberto, baseada na Web. O Hue agrupa vários projetos de ecossistemas do Hadoop diferentes numa única interface configurável, atuando como uma ferramenta de *front-end* para aplicações executadas no *cluster* (que seja mais "user-friendly").

A configuração do HUE envolve instalar a ferramenta num servidor dedicado, conectá-la aos serviços do Hadoop (como Hive, HBase, entre outros) e configurar numa base de dados *back-end* (geralmente MySQL ou PostgreSQL) para armazenar informação. Tem associado um editor SQL para consultas Hive e HBase; um File Browser para navegar no HDFS e um Job Browser para monitorizar os *jobs*. Por estas componentes torna-se uma ferramenta útil no ambiente de produção, pois permite ver todas as componentes adjacentes, numa interface amigável e intuitiva. Para otimização, é importante ajustar os limites de memória, conexões simultâneas e ativar *caching* de consultas frequentes.

O Hue suporta particionamento para otimizar consultas, replicação para garantir alta disponibilidade e consistência eventual para manter a integridade dos dados. Além disso, permite executar e monitorizar tarefas distribuídas, aproveitando a localidade dos dados e processamento paralelo.