MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Analyze and optimize the performance of Hive

Bachelor's Thesis

Author
Barnabas Maidics

Advisor
Peter Vary
Akos Dudas, PhD

October 19, 2018

Contents

HALLGATÓI NYILATKOZAT

Budapest, 2018. október 19.

Maidics Barnabas
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

1 Introduction

The digital era has led to large amounts of data being amassed by companies every day. Data comes from multiple sources: sensors, sales data, communication systems, logging of system events etc.. According to Forbes [1] 2.5 quintillion bytes of data created each day. That means 2.5 million Terabytes per day. Bigger corporations can easily create hundreds of Terabytes daily. So we need a new solution to process this amount of data. The traditional relational databases (RDBMS) can deal only with Gigabytes. Hadoop provides a software framework to scale up our system for storing, processing and analyzing big data.

In this chapter, I will write about the basics of Hadoop architecture, why Hive was created on top if it and the performance issues it faces.

1.1 Hadoop basics

Apache Hadoop is an open source distributed framework for managing, processing and storing a huge amount of data in clustered systems built from commodity hardware. All modules in Hadoop was designed with an assumption that hardware failures are common and should be automatically handled by the framework. One of the most important characteristic of Hadoop that it partitions the data and computation across many hosts and executing computation in parallel close to the data it uses. [5]

The base of the Hadoop framework contains the following modules:

- HDFS - Hadoop Distributed File System: designed to store large data sets reliably and stream those at high bandwidth to user applications.
- Hadoop MapReduce: an implementation of the MapReduce programming model for large data processing
- YARN - Yet Another Resource Negotiator: a resource management and job scheduling technology
- Hadoop Common: contains libraries and utilities needed by other Hadoop modules

1.1.1 Hadoop vs. traditional databases

Traditional databases cannot be used when we want to process and store big data. Main differences between Hadoop and traditional RDBMS:

- **Data Volume**: RDBMS works better when the data volume is low (Gigabytes). However when data size is huge (Terabytes-Petabytes) traditional databases fail. On the other hand Hadoop can easily handle this amount of data.
- **Data Variety**: this generally means the type of data to be processed. Hadoop has the ability to store and process data whether it is structured, semi-structured or unstructured. Even though it is mostly used for large amount of unstructured data. In contrast, traditional RDBMS can only be used to manage structured or semi-structured data.
- **Scalability**: RDBMS provides vertical scalability. You can add more resources, memory or CPU to a machine in the cluster. Whereas Hadoop provides horizontal scalability. It means we can add more machines to an existing cluster. As a result of this Hadoop becomes fault tolerant. We can easily recover data in case of a failure of one of the machines.
- **Data Processing**: Apache Hadoop supports OLAP (Online Analytical Processing) that involves very complex queries and aggregations. The database design is de-normalized, having fewer tables. On the other hand, RDBMS supports OLTP (Online Transaction Processing), which involves fast query processing. The database design is normalized having large number of tables. [4]

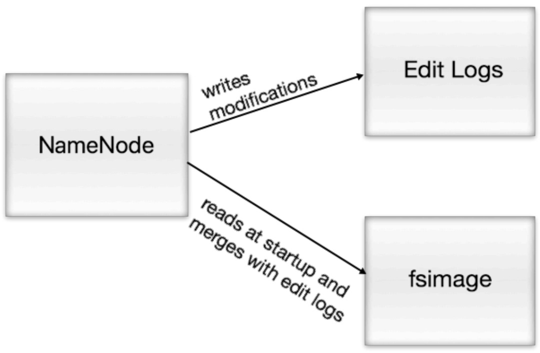
1.1.2 HDFS - Hadoop Distributed File System [3]

HDFS is the file system of Hadoop. It stores file system metadata and application data separately. The dedicated server that stores metadata is the NameNode. Application data are stored on other servers (DataNodes). These servers are connected and they communicate using TCP-based protocols.

NameNode [2]

NameNode keeps the directory tree of all files in the file system and tracks where file data is kept across the cluster. It does not store the files itself. Clients talk to the NameNode whenever they want to locate or add/move/copy/delete a file. The NameNode returns a list of relevant DataNode servers where the data is available.

As a result of this approach, the NameNode is a Single Point of Failure in the HDFS cluster. Whenever the NameNode goes down, the file system becomes offline.



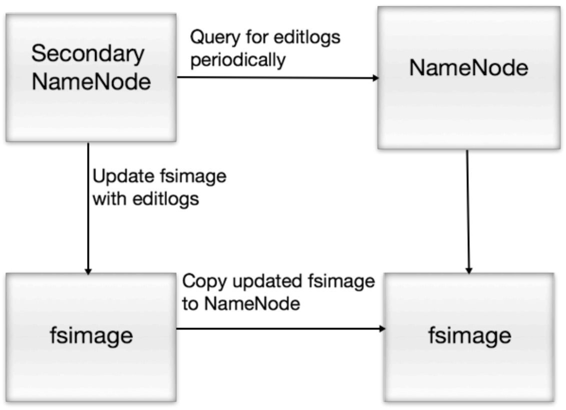
Problem with the NameNode

The image shows how NameNode stores information. There are two different files:

- edit logs: the changes made to the file system after the NameNode started
- fsimage: a snapshot of the file system when the NameNode started

In production clusters the NameNode restarts are very rare. That means edit logs can grow large therefore in case of a crash we will lose huge amount of metadata since the fsimage is very old.

The Secondary NameNode helps to address this issue. It takes over the responsibility of merging the edit logs with fsimage. It collects edit logs on a regular basis and applies them to the fsimage. NameNode will use this fsimage in case of a crash and it can also be used to reduce the startup time of the NameNode. It is important to remember that the Secondary NameNode is not a real backup NameNode it only merges the edits into the fsimage.


Secondary NameNode
Query for editlogs periodically
NameNode
Update fsimage with editlogs
fsimage
Copy updated fsimage to NameNode
fsimage

Solution using the Secondary NameNode

DataNodes

On a DataNode a block is represented by two files on the host's native file system. The first contains the data itself, the second is the metadata: checksum for the data and generation stamp.

On startup, the DataNodes connect to the NameNode and perform a handshake. The reason for the handshake, is to verify the namespace ID and software version of the DataNodes. If one of them does not match with the NameNode's value, the DataNode automatically shuts down. After a successful handshake the DataNode registers with the NameNode. DataNode will store it's internal identifier persistently. This way, if restart occurs the DataNodes will be recognizable even if they get a different IP address or port. After the ID is registered to the NameNode the ID of a DataNode will never change.

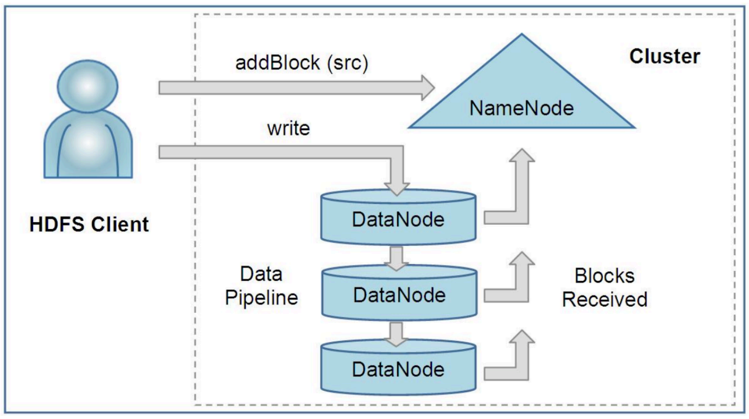
When a DataNode is registered it sends a block report immediately. It contains block id, generation stamp and the length of each block the DataNode server hosts. To provide up-to-date information to the NameNode reports are sent every hour.

During normal operation, DataNodes send heartbeats to the NameNode. It ensures the NameNode that the DataNode is operating and block replicas of the server are available. If the NameNode does not receive a heartbeat from a DataNode it will consider the node to be out of service. The default heartbeat interval is three seconds.

HDFS Client

User applications access the file system using the HDFS client. It exports the HDFS file system interface. Similar to a traditional file system, HDFS supports operations to read, write and delete files and to create or delete directories. The user references files and directories by paths in the namespace.

When someone reads a file, HDFS Client asks the NameNode for the list of DataNodes that host replicas of the blocks of the file. Then it will directly contacts the DataNode and request the desired block. When the client writes, it asks the NameNode to choose DataNodes to host replicas of the first block of the file. When the first block is filled, the client requests for new DataNodes to place the next blocks.



The client creates a new file by giving its path to the NameNode. For each block, the NameNode will return a list of DataNodes to place the replicas. The client pipelines data to the given DataNodes, and they will confirm the creation of the block to the NameNode.

1.1.3 MapReduce

1.1.4 Yarn

Acknowledgements

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Bibliography

[1] Forbes. How much data do we create every day. https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read.

[2] Madhukara Phatak. Secondary namenode. http://blog.madhukaraphatak.com/secondary-namenode---what-it-really-do/.

[3] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-1-4244-7152-2. DOI: 10.1109/MSST.2010.5496972. URL http://dx.doi.org/10.1109/MSST.2010.5496972.

[4] W3Training School. Hadoop vs rdbms. https://www.w3trainingschool.com/difference-big-data-hadoop-traditional-rdbms.

[5] Wikipedia. Apache hadoop. https://en.wikipedia.org/wiki/Apache_Hadoop.