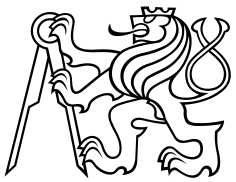
České vysoké učení technické v Praze
Fakulta stavební

Master thesis

Prague 2016

Matěj Krejčí

České vysoké učení technické v Praze
Fakulta stavební
Program Geodézie a kartografie
Obor Geoinformatika



Master thesis

Processing of vector data using distributed database systems in GIS

Využití distribuovaných databázových systémů pro správu vektorových dat v GIS

Vedoucí práce: Ing. Martin Landa, Ph.D.

Katedra geomatiky

Praha 2016

Matěj Krejčí

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta stavební
Thákurova 7, 166 29 Praha 6

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Krejčí Jméno: Matěj Osobní číslo: 384659

Zadávající katedra: Katedra geomatiky

Studijní program: Geodézie a kartografie

Studijní obor: Geomatika

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: Využití distribuovaných databázových systémů pro správu vektorových dat v GIS

Název diplomové práce anglicky: Processing of vector data using distributed database systems in GIS

Pokyny pro vypracování:
Diplomová práce se věnuje analýze technologií pro zpracování velkého množství dat (tzv. bigdata). Konktrétně je zaměřena na uložení a správu časoprostorových vektorových dat v prostředí distribuovaných databázových systémů jako je např. Hadoop. Cílem praktické části práce je jejich zpřístupnění a umožnění analýz v prostředí desktopového open source GIS nástroje GRASS GIS. Testování navrženého řešení bude prováděno s využitím virtualizované sítě uzlů tvořících cluster.

Seznam doporučené literatury:
Hadoop: The Definitive Guide O'Reilly Media, Inc. 2012 ISBN: 9781449311520
Programming Hive 1st O'Reilly Media, Inc. 2012 ISBN: 9781449319335
Big Data: Techniques and Technologies in Geoinformatics ISBN: 9781466586512

Jméno vedoucího diplomové práce: Ing. Martin Landa, Ph.D.

Datum zadání diplomové práce: 22.2.2016 Termín odevzdání diplomové práce: 22.5.2016

Podpis vedoucího práce Podpis vedoucího katedry

III. PŘEVZETÍ ZADÁNÍ

Beru na vědomí, že jsem povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v diplomové práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce“ a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací“.

Datum převzetí zadání Podpis studenta(ky)

Abstrakt

CIL..

Klíčová slova: GIS, GRASS

Abstract

Aim of...

Keywords: GIS, GRASS

Prohlášení

Prohlašuji že bakalářskou práci na téma „Analýza a vizualizace srážkových dat z mikrovlnných telekomunikačních spojů pomocí GIS“ jsem vypracoval samostatně. Veškerá použitá literatura je uvedena v seznamu zdrojů.

V Praze dne ....

(podpis autora)

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Ing. Martinu Landovi, Ph.D., za odbornou pomoc a rady, které byly často i nad rámec této práce. Můj dík patří také Ing. Martinu Fenclovi za pomoc a možnost spolupráce na projektu GACR 14-22978S. Rodině děkuji za příjemné a tvůrčí prostředí.

Obsah

Introduction

Context

Over the years the capabilities of hard drives have increased massively and access speed too. According to study by International Data Corp, since 2007 we produce more data than we can store. This amount of data is widely named big data.

Motivation and contribution

Aim of the thesis

Background of related work

Starter

1 Hadoop framework

Hadoop

"The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures."

History goes back to 2002 and the open-source project Apache Hadoop software for reliable, scalable, distributed computing was launched. The project was originally developed within Yahoo! company. That was the first company which used framework Hadoop in production environments.

At the beginning of October 2003, Hadoop project was launched to improve performance of the Apache Nutch web search engine in the short time (in January 2006) was moved to the new Hadoop sub project. At around the same time distributed file system called Google file system, with the specific, permitting efficient and reliable access the huge amount of data has been created. This abstract filesystem, widely called "user level" filesystem runs as a service that is accessible via APIs and

libraries. In 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community. By this time, Hadoop was being used by many other companies besides Yahoo!, such as Last.fm, Facebook, and the New York Times.

The base Apache Hadoop framework [**?**] written in Java is composed of the following modules:

- **Hadoop Common** - The common utilities that support the other Hadoop modules;
- **Hadoop Distributed File System (HDFS)** – A distributed file system that provides high-throughput access to application data;
- **Hadoop YARN** - A framework for job scheduling and cluster resource management;
- **Hadoop MapReduce** - an implementation of the MapReduce programming model for large scale data processing.

Beside main modules, there are many Hadoop extensions for cluster management, data access and helpers for storing data in HDFS.

1.1 HDFS: Hadoop Distributed File System

Hadoop comes with a filesystem and since it manages the storage of files across several machines, it is called Hadoop Distributed FileSystem (HDFS). Is designed for handling very large files with streaming data access. In HDFS, large files are broken down into smaller blocks (128MB, by default) which are stored as independent units. The architecture of HDFS is a highly fault-tolerant and provides wide permeability for access. A short overview of main characteristics of HDFS design is described below[**?**]:

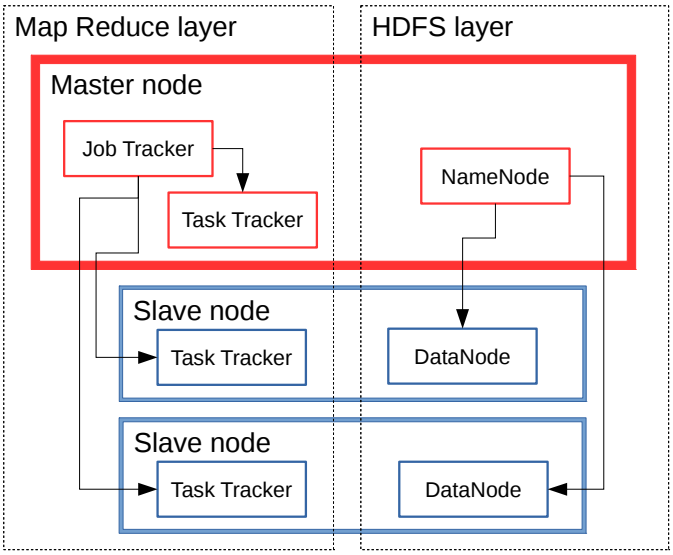
1. **Very large files** - With relevant hardware data of amounts petabites can be accessed on Hadoop clusters effectively.

2. **Streaming data access** - Efficient data processing is based on read once and copied many times pattern.

3. **Commodity hardware** is suitable for running Hadoop. At the end, this fact helped with decision to invest value of money to the development of Hadoop instead of operate on expensive and highly reliable hardware.

4. **Low-latency data access** - HDFS is not suitable for preforming low-latency access to data. Primary, HDFS is optimized for delivering a high throughput of data.

5. **Lots of small files** allows to read and operate over more files at the same time. Limitation of number of stored directories, files and block in filesystem is limited by *Namenode* memory.



Obr. 1.1: Hadoop architecture

Organisation of data

Similarly to a filesystem, HDFS as well is based on the disk blocks. The traditional the file system is based on blocks which define the minimal size of the amount of data to read and write. HDFS blocks have the simillar concept based on blocks, but the minimal unit is large. The size of the block is 128 MB by default. Blocks

are broken and distributed over disks on the cluster like blocks over a single disk in the filesystem. The ideal size of stored files is the same as the size of the block. With increasing the block size time cost for computing as well increasing. On the other hand, a large number of blocks is expensive for the preparation of files. The important task is to find optimize ratio between data preparation and computational time by set suitable blocks size. The idea of block abstraction helps to bring several benefits. First advantages of abstraction design is the possibility of storing bigger data file than physical disk unit, even the fact that it is unusual. The second benefit of fixed size is simplifying storage management. Obviously is easy task to hold calculate size disponibility and eliminating metadata concerns(Is not necessary to store metadata of tree in data blocks, even in the same system.) Furthmore the blocks are suitable for replication for providing fault tolerance. Protection against corrupted blocks, disk or machines is based replication of block over a cluster. To ensure the integrity of HDFS checksums when reading as well as reading data.

Namenodes

Hadoop comes with cluster architecture based on master(*Namenode*) and slave(*Datanode*) design pattern. The *Namenode* handles metadata of abstract file system(namespace), which include information about the structure of files and directories in the tree. Information about filesystem is stored on local disk in two files: the namespace image and edit log file. The *Namenode* know all locations of *Datanodes* blocks where data are stored. It also provides the primary user interface to access HDFS. By design *Namenode* is a single point of failure and should be never overloaded and must be the most reliable node of the cluster. Without *Namenode*, HDFS is totally unserviceable. In recent Hadoop releases, there is also a backupnode- *SecondaryNamenode*, always up to date with latest *Namenode* status. It receives all the operations done by *Namenode* and stores them in local memory. This permits to have the latest, up to date, namespace status when *Namenode* fails.

Datanodes

As has been mentioned, the blocks of a file are independently stored in nodes, which are called Datanodes. Every Datanode in the cluster makes registration pro-

cess to the Namenode during starting. Besides that, each Datanode informs Namenode about blocks availability by sending a block report. A block reports are sent periodically or when a change event happens. Moreover, every Datanode sends *relevant* messages to the Namenode to confirm that it remains operational and that the data is safe and available. If a Datanode stops operating, the error mechanisms designed to defend the failure and data loss maintain the availability of the block. *Relevant* messages also hold information, which allows the Namenode run the cluster efficiently e.g. load balancing. One important concept of design is that Namenode never directly calls data.

Data replication

I/O streams

1.2 Parallel computing - MapReduce

The *MapReduce* is a programming designed pattern for processing and generating large data sets. The *MapReduce* abstraction is inspired by the Map and Reduce functions, which is commonly found in functional programming languages, such as LISP. Users can easily express their computation as a series of Map and Reduce functions. The Map function processes a series of $<$ *key, value* $>$ pairs to generate a set of intermediate $<$ *key, value* $>$ pairs.

$$\text{Map(keyA, valueA)} \rightarrow \text{list (keyB, valueB)}$$

Reduce function aggregates all intermediate values that associate to the same intermediate key to produce the final output, also in the form of $<$ key, value $>$ pairs

$$\text{Reduce (keyB, list(valueB))} \rightarrow \text{list (key C, valueC)}$$

Thus the *MapReduce* framework transforms a list of (key, value) pairs into a list of values.

As MapReduce demonstration for this work is suitable to use we data from cellular microwaves links(MWL). Assume that data are already serialized and stored in text files. Each file representing 24 hours of captured data. Size amount of weakly data is relatively small(100Mb) which is suitable. We are interesting to compute mean of difference between transmitted and received signal for each link in period between 2014-07-07 and 2015-07-07.

Below is sample of data stored as CSV.

```
linkid;data;rx;tx
324;"2014-07-07 11:14:56.552";-48.9;10
256;"2014-07-07 11:14:59.703";-99.9;7
...
324;"2015-07-07 17:10:56.578";-50.1;7;
256;"2015-07-07 17:10:56.484";-85.3;10;
```

The lines above are presented to map functions as key-values pairs. The map function extracts data from date 2014-07-07 and creates pairs link:(rx-tx)

```
{324:-58.9}
{256:-106.9}
...
{324:-57.1}
{256:-95.3}
```

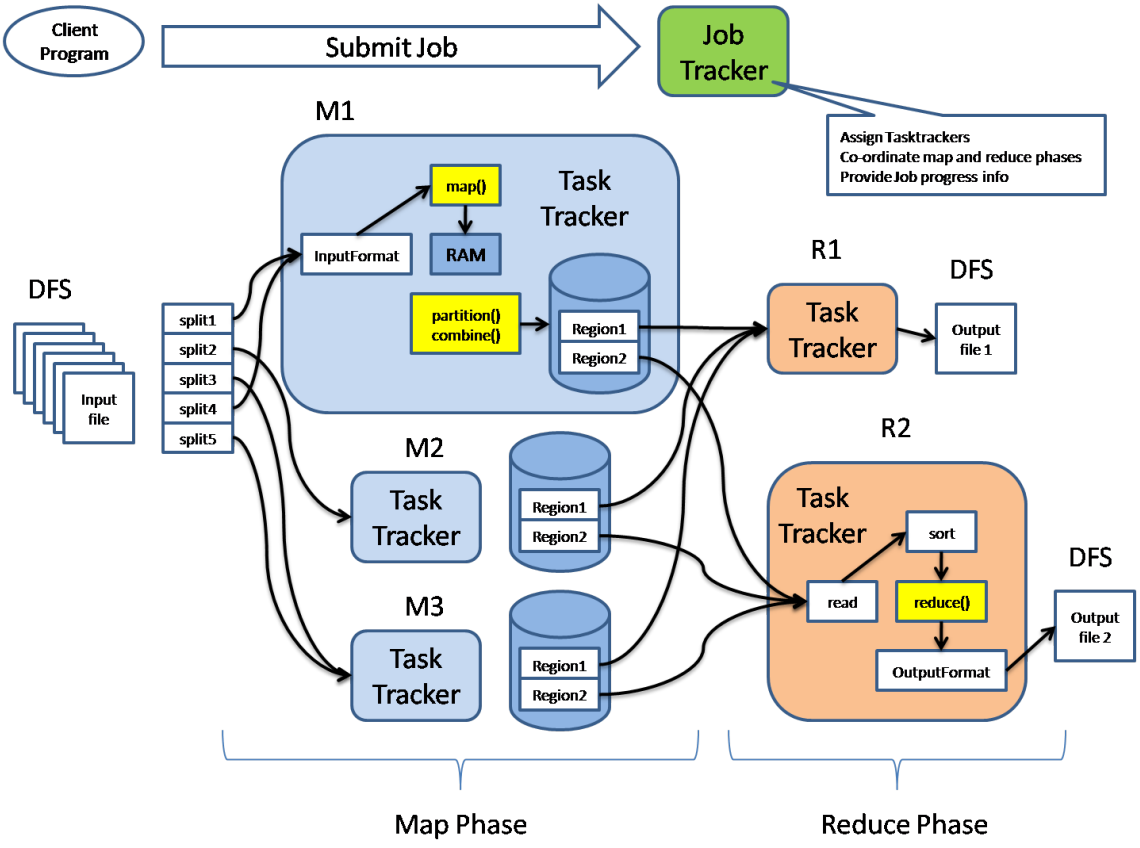
The output is processed by MapReduce framework before is being sent to reduce function. Mentioned process sorts pairs by key as bellow. Finally reduce program

```
{324:[-58.9,-57.1]}
{256:[-106.9,-95.3]}
```

iterate over the list of values for each link and compute mean. The final results are stored in Hadoop file system.

1.3 Execution workflow

The MapReduce framework splits the input files in to M pieces of typically 16 to 128 megabytes (MB) per piece. It then starts many copies of the program on a cluster, including 1 *Namenode* and other *Datanode*. The *Namenode* is responsible for scheduling the job to each *Datanode*, and monitoring the job progress and *Datanode's* health. *Datanode* with are asking to *Namenode* tasks of certain types (Map or Reduce, or both). The *Namenode* is checking the pool of non-running (including failed) tasks and assign one or more tasks to the Worker. Map tasks are assigned with priority to data locality. When a *Datanode* is assigned with a Map task, it first reads the content of the corresponding input split (either locally or remotely) and emits < key, value > pairs to the user-defined Map function. The output of the



Obr. 1.2: Running MapReduce job

Map function is first buffered in the memory and periodically written to local disks. A partitioning function partitions the output into R sets, each associated with a

Reducer task The *Namenode* passes the location of these outputs to *Datanode's* with Reduce tasks running, which read these buffered data. The Reduce task then sorts all the intermediate keys so that occurrences of the same key are grouped together. For each key, the entire list of values is passed to the Reduce function. Each Reducer, upon finishing its share of keys, outputs a result file (R output files in total).

2 Distributed spatial data

2.1 Serialization/De-serialization

2.2 Spatial indexing and analytic on Hadoop

2.3 Project: Spatial framework for Hadoop

Java geometry API

3 Technologies Behind

3.1 Spatial framework GRASS GIS

Postgis driver

3.2 Data warehouse: Hive

About

Performance limitation

3.3 Cluster virtualisation

Docker

Google cloud services

Experiment framework

Conclusion

Seznam použitých zkratek

API Application Programming Interface

CSV Comma Separated Values

EPSG Geodetic Parameter Set

GIS Geographic Information System

GNU GPL GNU General Public License

GRASS Geographical Resources Analysis Support System

GUI Graphical User Interface

Literatura

[1] RAGHAVAN, S. *Radar meteorology*. 31 Oct 2003. Boston: Kluwer Academic Publishers, 2003, 549 s. ISBN 14-020-1604-2.

Attachment