Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography
Vol. 35, No. 5, 405-414, 2017
https://doi.org/10.7848/ksgpc.2017.35.5.405

ISSN 1598-4850(Print)
ISSN 2288-260X(Online)
Original article

A Benchmark Test of Spatial Big Data Processing Tools and a MapReduce Application

Nguyen, Minh Hieu[1] · Ju, Sungha[2] · Ma, Jong Won[3] · Heo, Joon[4]

Abstract

Spatial data processing often poses challenges due to the unique characteristics of spatial data and this becomes more complex in spatial big data processing. Some tools have been developed and provided to users; however, they are not common for a regular user. This paper presents a benchmark test between two notable tools of spatial big data processing: GIS Tools for Hadoop and SpatialHadoop. At the same time, a MapReduce application is introduced to be used as a baseline to evaluate the effectiveness of two tools and to derive the impact of number of maps/reduces on the performance. By using these tools and New York taxi trajectory data, we perform a spatial data processing related to filtering the drop-off locations within Manhattan area. Thereby, the performance of these tools is observed with respect to increasing of data size and changing number of worker nodes. The results of this study are as follows 1) GIS Tools for Hadoop automatically creates a Quadtree index in each spatial processing. Therefore, the performance is improved significantly. However, users should be familiar with Java to handle this tool conveniently. 2) SpatialHadoop does not automatically create a spatial index for the data. As a result, its performance is much lower than GIS Tool for Hadoop on a same spatial processing. However, SpatialHadoop achieved the best result in terms of performing a range query. 3) The performance of our MapReduce application has increased four times after changing the number of reduces from 1 to 12.

Keywords: Spatial Big Data, Taxi Trajectory Data, Hadoop, Benchmark Test.

1. Introduction

Big Data processing has posed challenges due to increasing complexity of digital data in terms of types and quantity while the performance of the computer has reached a certain limit. Some organizations are providing big data processing services such as Cloudera (https://www.cloudera.com) or Hortonworks (https://hortonworks.com); however, these services still contain hindering factors such as cost and availability. Hadoop is a notable open source platform for big data processing released by Apache (https://apache.org), but it is still in development. Particularly, spatial big data poses unique statistical and computational challenges due to spatial data characteristics including dependency, spatial autocorrelation, anisotropy, heterogeneity, and multiscale and resolution (Jiang and Shekhar, 2017). Researchers and organizations around the world are attempting to overcome these challenges. Esri (http://www.esri.com) pioneered real-world problem solving by using geographic information systems. They published a spatial big data processing toolbox named GIS Tools for Hadoop which provides most of the basic functions dealing with spatial data processing. Source codes and additional information can be found on Github (https://github.com/Esri/gis-tools-for-hadoop). To

Received 2017. 9. 29, Revised 2017. 10. 12, Accepted 2017. 10. 31

1) Member, Dep. of Civil Engineering, Yonsei University (E-mail: hieuintelvn@gmail.com)
2) Dep. of Civil Engineering, Yonsei University (E-mail: jsh4907@yonsei.ac.kr)
3) Dep. of Civil Engineering, Yonsei University (E-mail: mjw1231@yonsei.ac.kr)
4) Corresponding Author, Member, Dep. of Civil Engineering, Yonsei University (E-mail: jheo@yonsei.ac.kr)

use GIS Tools for Hadoop, users need to install Hive (Thusoo *et al*., 2009) which is a data warehouse system for querying and analyzing large datasets in Hadoop. In academia, SpatialHadoop was released by Minnesota University (Eldawy and Mokbel, 2013). This tool inherits basic spatial data processing functions from the Esri Geometry API (Application Program Interface) library and uses Pig (https://pig.apache.org) to generate MapReduce applications. In SpatialHadoop, only some specific spatial operations have been developed in order to support researches which utilize k-NN to optimize filtering based on scope, or redefine the Pig Latin (Olston *et al*., 2008) to support spatial joints (Eldawy and Mokbel, 2015b). Besides, another tool is Hadoop-GIS was developed by Emory University (Aji *et al*., 2013) to deal with spatial big data processing. However, Hadoop-GIS only supports data up to two dimensional and two query types: rectangle range query and spatial joins (Garcia-Garcia *et al*., 2017). By evaluating the performance, the advantages and disadvantages of the tools can be figured out. Therefore, the regular users can choose the appropriate tool for their specific tasks. Researches on similar interests are listed: Performance evaluation of SpatialHadoop for big data was implemented in Isfahan University of Technology Isfahan, Iran (Maleki *et al*., 2016), Lawrence Berkeley National Laboratory conducted performance evaluation of a MongoDB and Hadoop platform for scientific data analysis (Dede *et al*., 2013), and Performance comparisons of spatial data processing techniques for a large-scale mobile phone dataset were implemented in University of Tokyo, Japan (Witayangkurn *et al*., 2012).

This paper presents an experiment to evaluate the performance of two tools: GIS Tools for Hadoop running on top of Hive and SpatialHadoop running on top of Pig. The experiment is carried out by implementing a spatial data processing related to filtering the drop-off locations within Manhattan area. Furthermore, a MapReduce application is introduced as a baseline to assess the results achieved by two tools. We tested this application with a different number of reduces to observe the changes in performance compared to using the default Hadoop parameters. Optimizing number of maps and reduces is not covered in this study.

2. Background

Hadoop: The Apache Hadoop is an open source software that provides reliable, scalable, and distributed computing. Hadoop has four main components: Hadoop common, Hadoop HDFS (Hadoop Distributed File System), Hadoop YARN (Yet Another Resource Negotiator), and Hadoop MapReduce. A basic structure of Hadoop is introduced in Fig. 1.

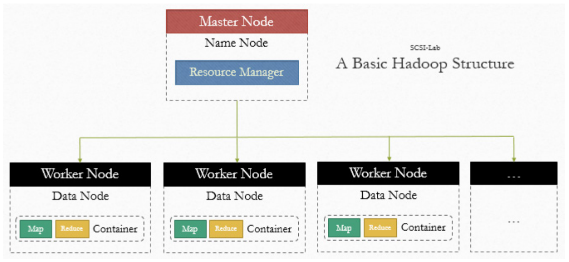


Fig. 1. A basic Hadoop structure

GIS Tools for Hadoop: GIS Tools for Hadoop is an open source toolkit that provides spatial analysis tools for big data(Jonathan, 2017). GIS Tools for Hadoop is composed 3 main components: Esri Geometry API for Java, Spatial Framework for Hadoop, and Geoprocessing Tools for Hadoop as shown in Fig. 2.

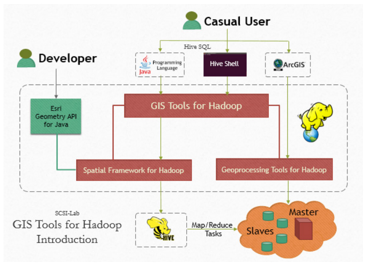
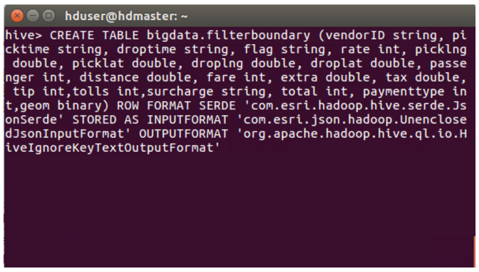

Casual User
Developer
Hive SQL
Java Programming Language
Hive Shell
ArcGIS
Esri Geometry API for Java
GIS Tools for Hadoop
Spatial Framework for Hadoop
Geoprocessing Tools for Hadoop
SCSI-Lab
GIS Tools for Hadoop Introduction
HIVE
Map/Reduce Tasks
Master
Slaves

Fig. 2. GIS Tools for Hadoop structure

- *Esri Geometry API for Java:* This library includes geometry objects, spatial operations, and spatial indexing (Quadtree). SpatialHadoop also used this library in the core spatial function processing (Whitman *et al*., 2014).
- *Spatial Framework for Hadoop:* This library includes user-defined functions (UDFs) that extends Hive and are built upon capabilities of the Esri Geometry API.
- *Geoprocessing Tools for Hadoop:* Using these tools, users



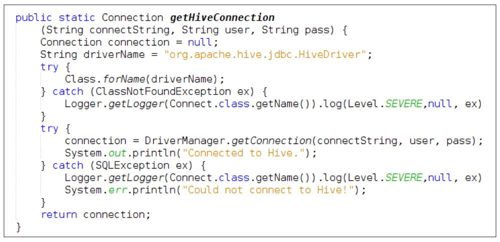
```
hive> CREATE TABLE bigdata.filterboundary (vendorID string, pi
cktime string, droptime string, flag string, rate int, picklng
 double, picklat double, droplng double, droplat double, passe
nger int, distance double, fare int, extra double, tax double,
 tip int,tolls int,surcharge string, total int, paymenttype in
t,geom binary) ROW FORMAT SERDE 'com.esri.hadoop.hive.serde.Js
onSerde' STORED AS INPUTFORMAT 'com.esri.json.hadoop.Unenclose
dJsonInputFormat' OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.H
iveIgnoreKeyTextOutputFormat'
```

Fig. 3. Submit a HiveSQL to Hive server through Hive shell

can connect data between Hadoop and ArcGIS, submit workflow jobs, and convert data to and from JSON.

Users can use GIS Tool for Hadoop following two options. Option 1: Entering the SQL statement directly from Hive shell window as shown in Fig. 3. In this option, the user will enter a sequence of statements to complete a task which is similar to this experiment. This approach is less user-friendly due to the re-entering of commands when an error occurs.

Option 2: Hive provides a protocol where the user can submit a sequence of commands through Java language to Hive server. To enable this feature, the user needs to start a second Hive server on the master node and create a connection before submitting SQL statements at the client side. Fig. 4 shows how the Hive connection was created by using Java. By this approach, users can save the entire process and debug the SQL statements through a Java-powered IDE (Integrated Development Environment) such as Netbeans (https://netbeans.org).



```java
public static Connection getHiveConnection
    (String connectString, String user, String pass) {
    Connection connection = null;
    String driverName = "org.apache.hive.jdbc.HiveDriver";
    try {
        Class.forName(driverName);
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(Connect.class.getName()).log(Level.SEVERE,null, ex)
    }
    try {
        connection = DriverManager.getConnection(connectString, user, pass);
        System.out.println("Connected to Hive.");
    } catch (SQLException ex) {
        Logger.getLogger(Connect.class.getName()).log(Level.SEVERE,null, ex)
        System.err.println("Could not connect to Hive!");
    }
    return connection;
}
```

Fig. 4. Create a Hive connection by Java

SpatialHadoop: SpatialHadoop is a comprehensive extension to Hadoop that injects spatial data awareness in each Hadoop layer, namely, the language, storage, MapReduce, and operations layers (Eldawy and Mokbel, 2015b). Fig. 5 describes the structure of SpatialHadoop.

When using SpatialHadoop, the user can choose one of two options. Option 1: Submitting commands directly from terminal console complying the syntax of SpatialHadoop. Option 2: Submitting a Pig Latin script from the terminal. In this experiment, we used option 2 to avoid re-entering the commands. Fig. 6 shows a Pig Latin script used in this experiment.

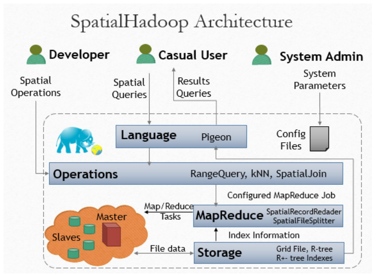
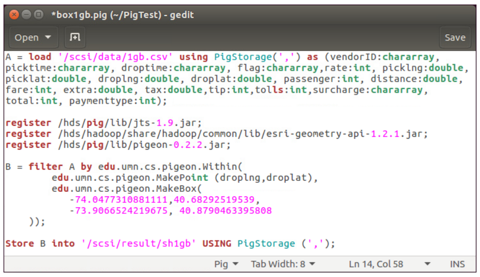


Fig. 5. Structure of SpatialHadoop



```
A = load '/scsi/data/1gb.csv' using PigStorage(',') as (vendorID:chararray,
picktime:chararray, droptime:chararray, flag:chararray,rate:int, picklng:double,
picklat:double, droplng:double, droplat:double, passenger:int, distance:double,
fare:int, extra:double, tax:double,tip:int,tolls:int,surcharge:chararray,
total:int, paymenttype:int);

register /hds/pig/lib/jts-1.9.jar;
register /hds/hadoop/share/hadoop/common/lib/esri-geometry-api-1.2.1.jar;
register /hds/pig/lib/pigeon-0.2.2.jar;

B = filter A by edu.umn.cs.pigeon.Within(
        edu.umn.cs.pigeon.MakePoint (droplng,droplat),
        edu.umn.cs.pigeon.MakeBox(
           -74.0477310881111,40.68292519539,
           -73.9066524219675, 40.8790463395808
    ));

Store B into '/scsi/result/sh1gb' USING PigStorage (',');
```

Fig. 6. Pig Latin script

Through the script, Pig directly reads the data from HDFS (Shvachko *et al*., 2010) and writes the result back to HDFS. This process is similar to how a MapReduce application works, but it works differently when compared to Hive. Hive organized and indexed data in directories which are stored in HDFS to help it works like a DBMS (Database Management System). Since Pig does not support spatial queries, SpatialHadoop has customized Pig by adding Pigeon (Eldawy and Mokbel, 2014) library.

So, users can manipulate these tools by entering single commands, entering a script or using Java. Eventually, the MapReduce applications are created and submitted to the Hadoop system. The differences between these tools are the number of maps and reduces created and how the data is processed in each map or reduce. By default, Hadoop determines the number of maps based on the number of DFS

(Distributed File System) blocks in the input file. If the user expects 10TB of input data and has 128MB DFS blocks, they will end up with 82000 maps (Apache, 2014). In our experiment, when using 0.5GB input data and 128MB DFS blocks (default), only 4 maps were created. Hive determines the number of maps in the same way of Hadoop while Pig cannot control map parallelism (Gates and Dai, 2016). In the reduce phase, if no setting is specified, Hadoop only creates one reduce. Hive and Pig determine the number of reduces based on the size of input data and 'hive.exec.reducers.bytes.per.reducer' or 'pig.exec.reducers.bytes.per.reducer' parameter. By default, if input data is 2GB, then 2 reduces will be requested by Hive or Pig. Sometimes, Pig does not implement the reduce phase, the whole process is done at the map phase (Gates *et al*., 2009). The number of maps and reduces can affect the performance since they can be run on the parallel mechanism (Fig. 7) by YARN (Vavilapalli *et al*., 2013). YARN is a component of Hadoop which creates virtual machines or containers on worker nodes to assign map or reduce tasks to them. A container can perform multiple map or reduce tasks, but if only one CPU (Central Processing Unit) is allocated to the container, the tasks will be performed sequentially. By default, YARN assigns only one core and 1GB of memory for each container. If a worker node has 4 cores CPU, only 4 tasks can be performed at the same time.

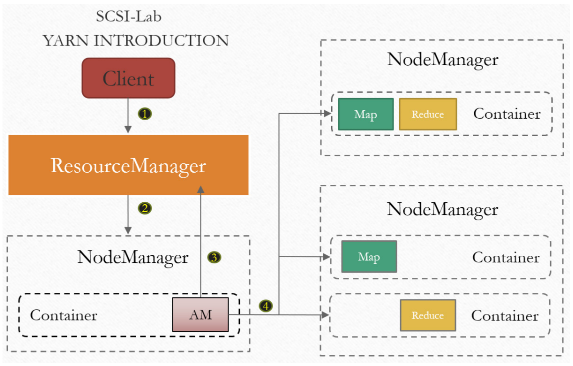


Fig. 7. MapReduce workflow of YARN

3. Experiment

3.1 Data set

The data used in this study is New York Taxi Trajectory data. The original data file is downloaded from NYC (the official website of the city of New York) Taxi and Limousine Commission (http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml) with the size 250MB for one month (Fig. 8). The data structure consists of 21 data fields, in which only 3 data fields (Table 1) are used for reading, and all fields are preserved during the writing the result. From this dataset, we created 3 new datasets to measure the performance by increasing data size: 0.5GB, 1GB, and 2GB.

Table 1. The fields used for data reading

Field Name	Description	Purpose
Dropoff_ longitude	Longitude where the meter was timed off.	Define Point Object
Dropoff_ latitude	Latitude where the meter was timed off.	Define Point Object
Dropoff_ datetime	The date and time when the meter was disengaged.	Define the key in MapReduce function

To obtain the Manhattan City boundary map, the US administrative unit data is downloaded from NYC in shapefile format (http://www1.nyc.gov/nyc-resources/agencies.page). This data is converted to WKT (http://www.opengeospatial.org/standards/wkt-crs) and GeoJSON (http://geojson.org) format since SpatialHadoop read a geometry object in WKT format and our MapReduce application read a geometry object in GeoJSON format. In order to provide another choice for the users in future, reading a geometry in GeoJSON format was tried.

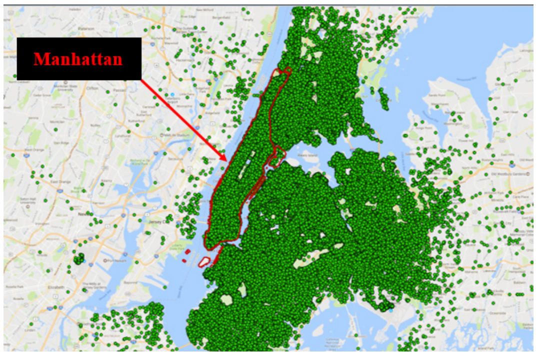


Fig. 8. New York taxi trajectory data

3.2. Cluster configuration

Number of nodes: 4 computers were used in this experiment and their information are described in Table 2. The most powerful computer was set as a master node, while the other computers with the same configuration were set as worker nodes. By using the same 3 computers for worker node, the performance comparison was conducted objectively with respect to increasing number of worker nodes.

Table 2. Hardware specifications

	Master Node (1)	Worker Node (3)
CPU	Intel Core i7-2600 @ 3.4GHz x 8	Intel Core i5-2300 @ 2.80Ghz x 4
Memory	16 GB	16 GB

Parameters setting: All tests are implemented with the default configuration of Hadoop, Hive, and Pig. Some parameters in Table 2 are used in result analysis.

Table 3. Some of default parameter setting

Parameter	Value	Description
Hadoop		
yarn.scheduler.minimum-allocation-vcores	1	The minimum allocation for every container request at the ResourceManager.
yarn.scheduler.minimum-allocation-mb	1024	The minimum allocation for every container request at the ResourceManager, in MBs.
mapreduce.job.reduces	1	The default number of reduce tasks per job.
dfs.block.size (MB)	128	Block size in HDFS. If your file is 1GB, Hadoop will create 8 blocks in HDFS.
Pig		
pig.exec.reducers.bytes.per.reducer (GB)	1	Defines the number of input bytes per reduce
mapred.min.split.size	-1	Sets the minimum split size defaults to dfs.blocksize
hive.exec.reduces.bytes.per.reducer (GB)	1	If the input size is 2G, Hive will request 2 reduces.

3.3. Implementation

In this study, we aim to filter out the taxi drop-off in Manhattan area (Fig. 9) from New York taxi trajectory data. To resolve the problem, three different tools were applied to process the filtering problem: SpatialHadoop(T1); GIS tools for Hadoop(T2); and MapReduce(T3). The workflow is illustrated in Fig. 10.

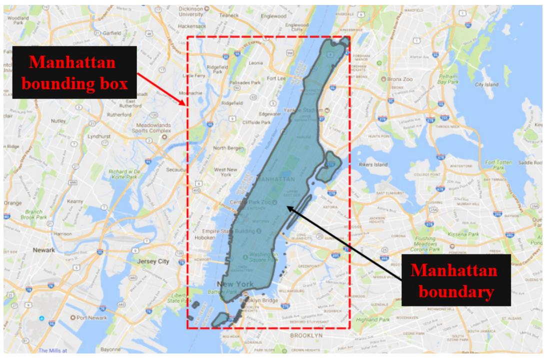


Fig. 9. Manhattan boundary and bounding box

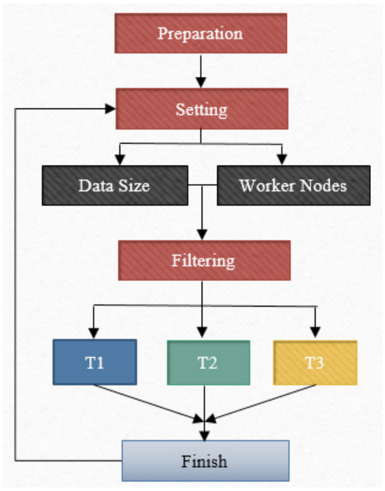


Fig. 10. Flowchart for experiment

'Point in Polygon' process can be implemented in one of following ways: 'contain' checking, 'within' checking, or 'intersect' checking. The impact of the checking algorithms on the performance test will not be covered in this study. We assume that using a same algorithm will not affect to the comparison of performance and chose 'within' checking to conduct our experiment. In MapReduce application, we used GeoTools (http://geotools.org) library which provided 'within' function for 'Point in Polygon' process. The input of this function consists of a point object (latitude, longitude) and a polygon object in GeoJSON format. The tools were tested based on the changing data size and changing number of worker nodes. For each test, repetition is necessary to ensure reliability. Besides, the impact of network factor is considered since Hadoop distributes tasks and aggregates results between nodes through network connection. However, this experiment was carried out on a small cluster (4 nodes) and LAN network, so the impact of network factor was neglected.

A spatial query on a large datasets are time-consuming and some studies have focused on building optimal solutions such as improving the performance of GIS polygon overlay computation with MapReduce for spatial big data processing (Wang *et al.*, 2015), High-performance spatial query processing on big taxi trip data using GPGPU (General-Purpose computing on Graphics Processing Units) (Zhang *et al.,* 2014), A spatial data partitioning framework for scalable query processing (Vo *et al.*, 2014). GIS Tools for Hadoop used a Quadtree index to speed up the performance (Whitman *et al.*, 2014), SpatialHadoop used two-level design to build a grid index, R-tree, and R+-tree (Eldawy and Mokbel, 2015a). However, creating the index in SpatialHadoop is not a simple task because the user needs to implement various steps such as using Java to customize the data format, compiling Java code in *.jar file format, and loading this file for every data reading/writing. Since this task is more suitable for a developer than for a regular user, the index was not created when using SpatialHadoop. Our MapReduce application is introduced as a baseline for this experiment without any data indexing strategy. Due to the mechanism of parallel data processing, the changing of performance can be observed when the number of maps and reduces are changed. Hadoop decides the number of maps based on input size and DFS block as mentioned. No specific number is given for the best number of reduces except the maximum number of 999 reduces. Therefore, some other tests on our MapReduce application with respect to changing number of reduces are implemented based on our research experience.

4. Evaluation

The performance of the tools were evaluated in three different scenarios: by changing the data size; by changing the number of worker node; and by changing tasks. The taxi drop-off filtering result is depicted in Fig. 11.

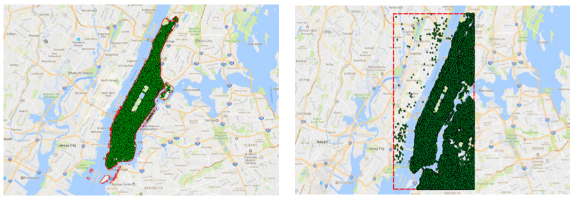


Fig. 11. Results after filtering process

4.1. Evaluation based on the changing data size

In Fig. 12, the processing time on T3 increased proportionally to the data size as no optimization was implemented. This is similar to T1 when the data size is increased from 0.5 to 1 GB; however, only a slight increase occurred when data size is increased from 1 to 2 GB. The time processing on T2 was remarkably lower than the other tools, however it slightly increased proportional to data size. The remarkable result can be deduced from leveraging the non-sequential data reading/writing mechanism of database management system from Hive and using Quadtree index in the spatial data processing.

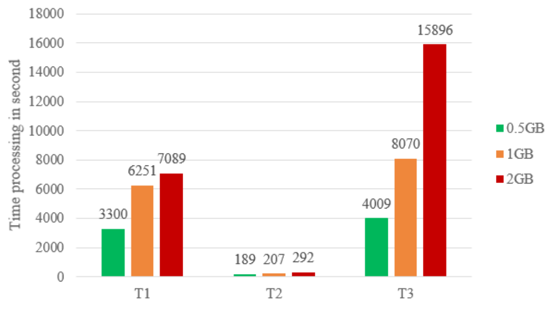


Fig. 12. The performance with respect to filtering by boundary, increasing data size, and using 2 worker nodes

4.2. Evaluation based on the changing number of worker nodes

Fig. 13 shows that when testing with 0.5GB data, the number of nodes almost did not affect the performance. In other words, this amount of data is not large enough to affect the performance of a Hadoop cluster. Because when using 0.5GB, only 4 maps and 1 reduce were created at max. Our worker node has 4 cores and 16GB memory, so the job is always done with only one worker node specified. This test was repeated on 1GB data and Fig. 14 shows that when the number of worker nodes is increased from 1 to 2, the changing performance on T1 and T2 is noticeable. This can be explained as when loading 1GB data, there are always 8 maps created in map phase. These maps can be performed at the same time with 8 cores and 8 GB of memory (2 nodes contain 8 cores and 32 GB of memory). At the reduce phase, Pig did not implement any reduce task in this case (Fig. 15). In other words, the whole process was done at the map phase (8 map tasks). So, the performance has increased when compared to using only one node with 4 map tasks created. T2 created two reduce tasks due to the fact that our data file is 1031183828 byte (> 1000000000 byte or 1GB). With 2 reduces running at the same time, the performance of T2 was improved. Similarly, there is no improvement when using 1GB data and increasing the number of worker nodes from 2 to 3.

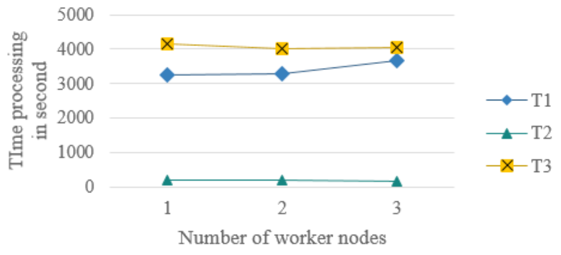


Fig. 13. The performance with respect to filtering by boundary, increasing the number of worker nodes, and using 0.5GB data

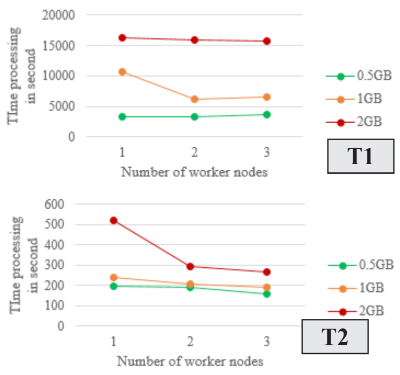
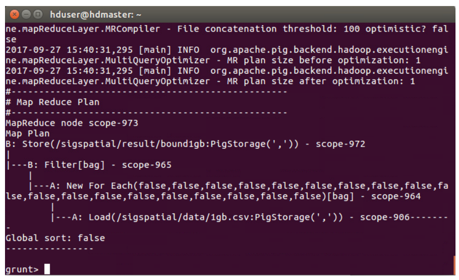


Fig. 14. The performance on T1 and T2 with respect to filtering by boundary, increasing the number of worker nodes, and increasing the data size



```
hduser@hdmaster: ~
ne.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? fal
se
2017-09-27 15:40:31,295 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1
2017-09-27 15:40:31,295 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1
#--------------------------------------------------
# Map Reduce Plan
#--------------------------------------------------
MapReduce node scope-973
Map Plan
B: Store(/sigspatial/result/bound1gb:PigStorage(',')) - scope-972
|
|---B: Filter[bag] - scope-965
    |
    |---A: New For Each(false,false,false,false,false,false,false,false,false,fa
lse,false,false,false,false,false,false,false,false,false)[bag] - scope-964
        |
        |---A: Load(/sigspatial/data/1gb.csv:PigStorage(',')) - scope-906-------
-
Global sort: false
----------------

grunt> 
```

Fig. 15. Map Reduce plan on T1 after optimization by Pig optimizer

On T3, since there is always one reduce task created, the performance did not changed when the number of worker nodes are changed. Fig. 16 describes clearly this inadequacy on the T3.

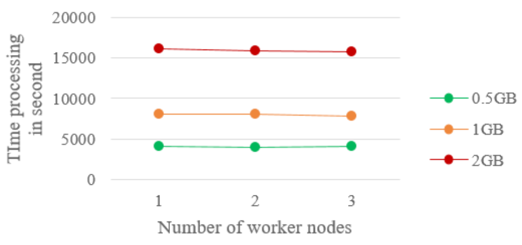


Fig. 16. The performance on T3 with respect to filtering by boundary, increasing the number of worker nodes, and increasing the data size

From the inadequacy mentioned, an adjustment was applied on T3 by changing the number of reduce tasks. In our experience, the number of reduce tasks depends on the outcome of the map phase and the resources of Hadoop cluster. If too many reduces are created, the output file will be split into many small parts which are equivalent to the number of reductions created. This is detrimental to subsequent processing. The purpose of this test is to clarify that the user should not use all default Hadoop parameters. Fig. 17 showed that the performance on T3 has improved significantly as the number of reduce tasks was altered from 1 to 12.

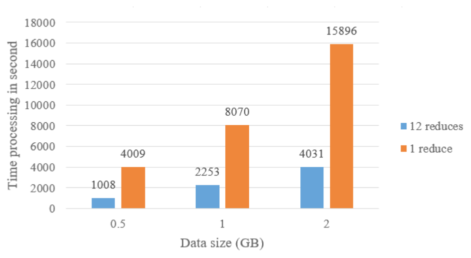


Fig. 17. The performance on T3 with respect to filtering by boundary, increasing the data size, and changing number of reduces

4.3. Evaluation based on the changing task

In fact, filtering taxi drop-off location by Manhattan bounding box can be done through a range query by comparing coordinates. In this case, T1 has achieved the best performance (Fig. 18) as using SpatialFileSplitter (Eldawy and Mokbel, 2015b) with a range filter to select the blocks overlapping with the query area, while T2 achieved excellent performance with respect to filtering by Manhattan boundary (Fig. 19).

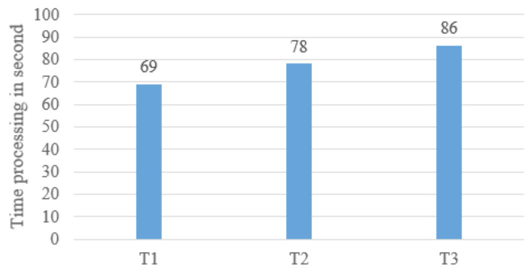


Fig. 18. The performance with respect to filtering by bounding box and using 2 worker nodes

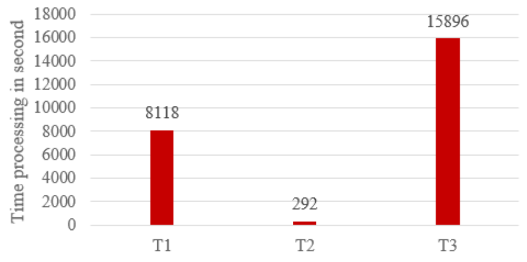


Fig. 19. The performance with respect to filtering by boundary and using 2 worker nodes

5. Conclusion

This paper presented a benchmark test between two notable tools of spatial big data processing: GIS Tools for Hadoop and SpatialHadoop with respect to filtering taxi drop-off location within Manhattan area, while increasing data size and changing number of worker nodes. GIS Tools for Hadoop is superior to SpatialHadoop as it automatically generates a Quadtree index in each spatial query. In addition, GIS Tools for Hadoop leveraged Hive's advantages to speed up data access in HDFS. Therefore, its performance has improved significantly. However, users should be familiar Java to handle this tool conveniently. SpatialHadoop supported an R-tree index but users must create this index manually through various steps. In the test of filtering by Manhattan boundary, the performance of SpatialHadoop is much lower than GIS Tool for Hadoop as no index was used. In the case of filtering by Manhattan bounding box, SpatialHadoop achieved the best performance in term of performing an optimal range query. Therefore, in some cases of estimation or preliminary processing, users can use SpatialHadoop as a second option. The number of maps and reduces affected to the performance of a MapReduce application. Using only one reduce was not effective. Thus, users should alter the number of reduce tasks flexibly to improve the performance. Future work will test the performance on indexed data and measure the difference between IO disk and network performance. Further, a framework for parallel processing of taxi trajectory data can be developed.

Acknowledgment

This research, 'Geospatial Big Data Management, Analysis and Service Platform Technology Development', was supported by the MOLIT(The Ministry of Land, Infrastructure, and Transport), Korea, under the national spatial information research program supervised by the KAIA(Korea Agency for Infrastructure Technology Advan cement)"(17NSIP-B081011-04).

References

Aji, A., Sun, X., Vo, H., Liu, Q., Lee, R., Zhang, X., and Wang, F. (2013), Demonstration of Hadoop-GIS: a spatial data warehousing system over MapReduce, *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM, 05-08 November, Orlando, USA, pp. 528-531.

Apache. (2014), How many maps and reduces, Apache, Wakefield, USA, https://wiki.apache.org/hadoop/HowManyMapsAndReduces (last date accessed: 15 October 2017).

Dede, E., Govindaraju, M., Gunter, D., Canon, R. S., and Ramakrishnan, L. (2013), Performance evaluation of a Mongodb and Hadoop platform for scientific data analysis, *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing* 2013, ACM, 17 June, New York, USA, pp. 13-20.

Eldawy, A. and Mokbel, M. F. (2013), A demonstration of SpatialHadoop: an efficient mapreduce framework for spatial data, *Proceedings of the VLDB Endowment*, VLDB, 26-30 August, Riva del Garda, Italy, Vol. 06, No. 12, pp. 1230-1233.

Eldawy, A. and Mokbel, M. F. (2014), Pigeon: a spatial MapReduce language, *Proceedings of 30th International Conference on Data Engineering (ICDE)* 2014, IEEE, 31 March - 04 April, Chicago, USA, pp. 1242-1245.

Eldawy, A. and Mokbel, M. F. (2015a), The ecosystem of SpatialHadoop, *Proceedings of SIGSPATIAL Special*, ACM, 03-06 November, Seattle, USA, Vol. 06, Issue 03, pp. 03-10.

Eldawy, A. and Mokbel, M. F. (2015b), SpatialHadoop: A MapReduce framework for spatial data, *Proceedings of 31st International Conference on Data Engineering (ICDE)* 2015, IEEE, 13-17 April, Seoul, Korea, pp. 1352-1363.

Garcia-Garcia, F., Corral, A., Iribarne, L., Mavrommatis, G., and Vassilakopoulos, M. (2017), A comparison of distributed spatial data management systems for processing distance join queries, In: Kirikova, M., Nørvåg, K., and Papadopoulos, G. (eds.), *Advances in Databases and Information Systems*, Springer, Cham, Switzerland, pp. 214-228.

Gates, A. and Dai, D. (2016), *Programming Pig: Dataflow Scripting with Hadoop*, O'Reilly Media, Sebastopol, USA, pp. 65-66.

Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayanamurthy, S. M., Olston, C., and Srivastava, U. (2009), Building a high-level dataflow system on top of MapReduce: the Pig experience, *Proceedings of the VLDB Endowment*, VLDB, 24-28 August, Lyon, France, Vol. 02, pp. 1414-1425.

Jiang, Z. and Shekhar, S. (2017), *Spatial Big Data Science*, Springer, Cham, Switzerland, pp. 03-13.

Jonathan, M. (2017), GIS tools for Hadoop, Esri, Readlands, USA, https://blogs.esri.com/esri/arcgis/2013/03/25/gis-tools-for-hadoop (last date accessed: 17 October 2017).

Maleki, E. F., Azadani, M. N., and Ghadiri, N. (2016), Performance evaluation of SpatialHadoop for big web mapping data, *Proceedings of 2nd International Conference on Web Research (ICWR)*, IEEE, 27-28 April, Tehran, Iran, pp. 60-65.

Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008), Pig latin: a not-so-foreign language for data processing, *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ACM, 09-12 June, Vancouver, Canada, pp. 1099-1110.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The Hadoop distributed file system, *Proceedings of 26th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, 03-07 May, Incline Vilage, USA, pp. 01-10.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. (2009), Hive: a warehousing solution over a map-reduce framework, *Proceedings of the VLDB Endowment*, VLDB, 24-28 August, Lyon, France, Vol. 02, pp. 1626-1629.

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., and Saha, B. (2013), Apache Hadoop YARN: yet another resource negotiator, *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC)*, ACM, 01-03 October, Santa Clara, USA, pp. 05-10.

Vo, H., Aji, A., and Wang, F. (2014), A spatial data partitioning framework for scalable query processing, *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM, 04-07 November, Dallas/Forth Worth, USA, pp. 545-548.

Wang, Y., Liu, Z., Liao, H., and Li, C. (2015), Improving the performance of GIS polygon overlay computation with MapReduce for spatial big data processing, *Cluster Computing Journal,* Vol. 18, Issue 02, pp. 507-516.

Whitman, R. T., Park, M. B., Ambrose, S. M., and Hoel, E. G. (2014), Spatial indexing and analytics on Hadoop, *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM, 04-07 November, Dallas/Forth Worth, USA, pp. 73-82.

Witayangkurn, A., Horanont, T., and Shibasaki, R. (2012), Performance comparisons of spatial data processing techniques for a large scale mobile phone dataset, *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications*, ACM, 01-03 July, Reston, USA, pp. 25-31.

Zhang, J., You, S., and Gruenwald, L. (2014), High-performance spatial query processing on big taxi trip data using gpgpus, *Proceedings of International Congress on Big Data*, IEEE, 27-30 October, Washington, USA, pp. 72-79.