Speeding DBLP querying using hadoop and spark

M Alsmirat, Y Jararweh and M Al-Ayyoub
Jordan University of Science and Technology, Computer and Information Technology, Computer Science Department, Jordan
E-mail: masmirat@just.edu.jo

Abstract. Big data is becoming bigger every day. Even for simple applications such as the Digital Bibliography & Library Project (DBLP) database, the data is becoming unmanageable using the conventional databases because of its size. Applying big data processing methods such as Hadoop and Spark is becoming more popular because of that. In this work, we investigate the use of Hadoop and Spark in the querying process of big data and we compare the performance of them in terms of their execution time. We use the DBLP database as a case study. Results show that Hadoop and Spark enhances the query execution time significantly when compared with conventional database management systems. We also found that Spark enhances the execution time over Hadoop.

1. Introduction
Big data is a term for data sets that are so large or complex for traditional data processing application software to deal with them [1]. Relational database management systems and desktop statistics and visualization packages often have difficulty handling big data. This processing may require parallel software running on maybe thousands of servers. Big data solutions can be classified according to the approach used in storing and processing these data. The first approach is the traditional enterprise approach where an enterprise have a mainframe computer to store and process big data. The second approach is the MapReduce algorithm [2], which was proposed by Google. MapReduce divides the task into small parts and assigns them to many computers, and collects the results from them which when integrated, form the result dataset. Apache implemented the MapReduce approach into Apache Hadoop [3] and Apache Spark [4]. DBLP is a computer science bibliography website which index all important journals and conferences on computer science. In this paper, we loaded the DBLP dataset [5] in the aforementioned two approaches. More specifically, we used Oracle 12c to implement the first approach and we used Apache Hadoop with Apache Hive [6] and Apache SparkeSql [7] to implement the second approach. We compare the execution time of querying the DBLP dataset using the three approaches.

2. Background Information

2.1. Apache Hadoop with Apache Hive
Hadoop is an open source framework by Apache that is implemented in java. This framework allows decentralized processing of large datasets over distributed computers. The Hadoop framework also provides distributed storage to be able to facilitate its distributed computation. Furthermore, Hadoop is designed to be scalable in a way that it can be run on a single machine or
thousands of machines which offer its own computation power and storage. Hadoop framework consists of the following two major layers.

(i) Processing/Computation layer (MapReduce)
(ii) Storage layer (Hadoop Distributed File System).

Hadoop does not have internal data warehouse. Usually, use Apache Hive is used as Hadoop data warehouse. Hive is a tool that is usually used to be able to process structured data in Hadoop. Hive can be seen as a layer above Hadoop facilitate querying and analyzing Hadoop data easy. Initially, Hive was developed in Facebook. From there, Apache adapted it and developed it as an open source with the name Apache Hive. Currently, Hive is being used by different companies such as Amazon in Amazon Elastic MapReduce. Hive provides an SQL like language, called HiveQL or HQL, for querying data. Figure 1 shows the Hive internal architecture.

![Hive Architecture](https://www.tutorialspoint.com/hive/hive_introduction.htm)

**Figure 1.** Hive Architecture

### 2.2. Apache Spark

Apache Spark came to speed the processing time of Hadoop and to efficiently use more types of operations. These operation may include interactive queries and stream processing. In these operations, the main concern is to maintain high speed in processing large datasets to make the waiting time between queries and the waiting time to run a program very short. As Spark has its own computation distribution manager, it can be used without Hadoop. With that said, Spark still can be implemented using Hadoop. Spark may use Hadoop for storage or for processing. Spark main components are shown in Figure 2.

As Spark has its own internal SQL components, it does not need any warehouse to be install with.

### 3. Materials and Methods

We installed Hadoop and Hive on a laptop that has 5th generation quad core i7, 16 GB or RAM, and an NVIDIA GTX 960M. We then downloaded the DBLP database as an XML file. The file contains around 5 million records.

To use Hadoop with Hive on a single machine, it should be first installed on the machine and then Hive is installed above it. After that, Hive should be run using the command hive on a terminal. After that, to convert any XML data you want to use to Hive required format, an

1 [https://www.tutorialspoint.com/hive/hive_introduction.htm](https://www.tutorialspoint.com/hive/hive_introduction.htm)
2 [https://intellipaat.com/tutorial/spark-tutorial/apache-spark-components/](https://intellipaat.com/tutorial/spark-tutorial/apache-spark-components/)
external tool called XmlSerDe \(^3\) that is implemented as a jar file can be used. In our case, the DBLP dataset was converted, loaded, and processed using the following code.

```sql
CREATE EXTERNAL TABLE PUBLICATIONS
(author array<STRING>, title STRING, journal STRING)
ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe'
WITH SERDEPROPERTIES
("column.xpath.author"="/article/author/text()",
"column.xpath.title"="/article/title/text()",
"column.xpath.journal"="/article/journal/text()"
) STORED AS INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
LOCATION '/test/book6'
TBLPROPERTIES ("xmlinput.start"="<article","xmlinput.end"= "</article>");
```

After running the previous code, SQL queries can be used to query the data. In this paper, we used the queries in Figures 3, 4, and 5.

To use We use eclipse IDE with its APIs to run our code on it. A Maven project is created and pom.xml (the dependencies XML file) is edited to configure Spark and SparkSQL.

```xml
<dependencies>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.10</artifactId>
    <version>2.0.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_2.10</artifactId>
    <version>2.0.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.10</artifactId>
    <version>2.0.2</version>
  </dependency>
</dependencies>
```

\(^3\) https://github.com/dvasilen/Hive-XML-SerDe
The following Scala language script is written to load the dataset and be able to run the same queries in Figures 3, 4, and 5 on it.

```scala
package com.spark_example
import org.apache.spark._
import org.apache.spark.{SparkConf , SparkContext}
import org.apache.spark.sql.{DataFrame , SQLContext}
import com.databricks.spark.xml
import javafx.scene.input.DataFormat
import org.codehaus.jackson.format.DataFormatDetector
import org.apache.spark.sql.DataFrameReader
object main{
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("PUBLICATIONS").setMaster("local")
    val sc = new SparkContext(conf)
    val sqlContext : SQLContext = new org.apache.spark.sql.SQLContext(sc)
    loadBookData(sqlContext)
  }
  def loadBookData(sqlContext : SQLContext){
    var df:DataFrame = null
    var newDF:DataFrame = null
    import sqlContext.implicits._
    df = sqlContext.read.format("xml").option("rowTag", "article").load("dblp.xml")
    df.registerTempTable("dblp")
    sqlContext.sql("SELECT author FROM dblp where journal = 'Acta Inf.' and year = '2000'").show()
  }
}
```

Figure 3. The SQL Query 1.
4. Experiments and Results
On the three implementations, we run the queries shown in Figures 3, 4, and 5 each for ten times and we measure the execution time of each run. We then calculate the average of the 10 runs of each query on each implementation. The results are shown in Tables 1, 2, and 3. As can be seen clearly from the results, the Spark implementation outperformed all other implementations significantly. It reduces the execution time by almost 2X when compared with Hadoop and by

Table 1. Execution time of different implementations for Query 1

| Implementation | Average Execution time (Seconds) |
|----------------|----------------------------------|
| Oracle         | 10821.03133                      |
| Hadoop         | 792.4905                         |
| Spark          | 363.2                            |

Table 2. Execution time of different implementations for Query 2

| Implementation | Average Execution time (Seconds) |
|----------------|----------------------------------|
| Oracle         | 16336.938                        |
| Hadoop         | 1014.6898                        |
| Spark          | 503.3                            |

Table 3. Execution time of different implementations for Query 3

| Implementation | Average Execution time (Seconds) |
|----------------|----------------------------------|
| Oracle         | 10825.19267                      |
| Hadoop         | 568.6398                         |
| Spark          | 503.3                            |
almost 32X when compared with Oracle.

References
[1] Hilbert M and Lopez P 2011 science 1200970
[2] Dean J and Ghemawat S 2008 Commun. ACM 51 107–113 ISSN 0001-0782 URL http://doi.acm.org/10.1145/1327452.1327492
[3] Apache Hadoop URL http://hadoop.apache.org/
[4] Apache Spark URL https://spark.apache.org/
[5] DBLP dataset URL https://dblp.uni-trier.de/faq/How+can+I+download+the+whole+dblp+dataset
[6] Apache Hive URL https://hive.apache.org/
[7] Apache Spark sql URL https://spark.apache.org/sql/