ISSN No. 0976-5697

Volume 3, No. 7, Nov-Dec 2012

International Journal of Advanced Research in Computer Science

CASE STUDY AND REPORTS

Available Online at www.ijarcs.info

Using Pig on Hadoop for Data Analysis in Bioinformatics

Smita Saxena
Bioinformatics Centre
University of Pune Pune, India
smita@bioinfo.net.in

Abstract: Data storage, processing and analysis is a major component of bioinformatics. Apache Hadoop provides a distributed computing framework for processing large voluminous data. Pig is an Apache open source project that works on the Hadoop platform and let the programmer write the queries or scripts in its procedural dataflow language known as Pig Latin rather than writing core MapReduce programs in Java directly. Pig provides a lot of statements similar to SQL clauses and some other advanced features. The Pig platform compiles the statements and scripts and generates the equivalent map and reduce tasks and sends to Hadoop for execution. It helps to process the biological data available in large sizes, which can be analyzed in an effective way within a small time. Also in contrast to traditional (R)DBMS, unconstrained data may be used and the database schema also need not to be constrained or consistent or pre-defined.

Keywords: Bioinformatics, Hadoop, Pig, Pig Latin.

I. INTRODUCTION

Bioinformatics is a computational approach to manage, store and analyze the large biological data, which includes protein and nucleotide sequences, macromolecule structures and other functional experiments results. Apart from just the genomic and proteomic sequences, there is a vast amount of biochemical information, metabolic pathways, regulatory networks, protein-protein interaction data, phylogenetic information etc [1]. The next generation high throughput experiments typically known as NGS also generates a lot of data by parallelizing the sequencing process and produces thousands of sequences at once at a relatively low cost [2].

To deal with this enormous amount of data, Hadoop® framework [3] provides a platform for the large-scale data analysis [4]. Hadoop® project developed and maintained by Apache Software Foundation™ is a open-source software for reliable, scalable, distributed computing available at http://www.hadoop.apache.org/. It includes three subprojects: a. *Hadoop Distributed File System*™ (HDFS) - the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable, extremely rapid computations. b. *Hadoop MapReduce* - a software framework for distributed processing of large data sets on compute clusters. c. *Hadoop Common* - The common utilities for File System, RPC and serialization libraries that support the other Hadoop subprojects (some of which have now become top-level Apache projects) as shown in table I.

Table I. Hadoop Related Other Apache Projects

Project	URL & Description
Avro™	(http://www.avro.com) A data serialization system.
Bigtop™	(http://incubator.apache.org/bigtop/) A project for the development of packaging and tests of the Apache Hadoop ecosystem.
Cassandra ™	(http://cassandra.apache.org/)A scalable multi-master database with no single points of failure.
Chukwa™	(http://incubator.apache.org/chukwa/) A data collection system for monitoring large distributed systems.
Flume™	(http://incubator.apache.org/flume/)A distributed, reliable, and available system for efficiently collecting, aggregating, and moving large amounts of log data from many different sources to a centralized data store.
Hama™	(http://hama.apache.org/) A pure BSP (Bulk Synchronous Parallel) computing framework on top of HDFS for massive scientific computations such as matrix, graph and network algorithms.
Hbase™	(http://hbase.apache.org/) A scalable, distributed database that supports structured data storage for large tables.
Hive™	(http://hive.apache.org/) A data warehouse infrastructure that provides data summarization and ad hoc querying.
Mahout™	(http://mahout.apache.org/) A Scalable machine learning and data-mining library.
Oozie™	(http://incubator.apache.org/oozie/) A workflow/coordination system to manage Apache Hadoop jobs.
Pig™	(http://pig.apache.org/) A high-level data-flow language and execution framework for parallel computation.
Sqoop™	(http://sqoop.apache.org/) A tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.
Whirr™	(http://whirr.apache.org/) A Library for deploying and running Hadoop in the cloud.
Zookeeper ™	(http://zookeeper.apache.org/) A high-performance coordination service for distributed applications.

II. HDFS AND MAPREDUCE PROCESS

HDFS and MapReduce are based on Google's MapReduce and File System specification [5]-[7]. Hadoop provides an implementation of the MapReduce framework written in Java and released under a free licence [8]. A number of other open-source Hadoop distribution are also available like GreenPlum™ HD(http://greenplum.com/ products/greenplum-hd), CDH- Cloudera© Hadoop distribution (http://www.cloudera.com/hadoop/), IBM® InfoSphere BigInsights (http://www-01.ibm.com/software/ data/infosphere/biginsights/) etc.

Hadoop File System HDFS is highly fault tolerant and designed to support large files. There are two types of HDFS nodes in a Hadoop cluster- *NameNode* which manages file system namespace and metadata, and *DataNodes* which store blocks of data. A block of data is typically 64MB or 128MB in size and is replicated on many nodes of the cluster; default is to replicate every block three times. There are two types of MapReduce nodes- *JobTracker* and *TaskTracker*. JobTracker is only one and manages MapReduce jobs; it receives the jobs submitted and schedules Map and Reduce tasks on TaskTracker and monitors them. Each TaskTracker spawns JVM (Java Virtual Machine) to run map and reduce task.

A MapReduce program consists of a map function and a reduce function. A MR job is an executing MR program that is divided into map tasks that run in parallel with each other and reduce tasks that run in parallel with each other. A "map" function transforms each piece of data into some number of key/value pairs. Each of these elements will then be sorted by their key and reach to the same node, where a "reduce" function is used to merge the values (of the same key) into a single result [5].

map (k1,v1) => list(k2,v2) =>
reduce (k2,list(v2)) => list(v2)

The Map and Reduce functions have to be implemented in accordance with the problem requirement.

III. THE PIG PROJECT

Pig^{TM} started as a research project in Yahoo! Research (http://developer.yahoo.com/hadoop/) to focus more on analyzing large data sets and spend less time in writing MapReduce programs. It was designed as an in-between of the declararative style of SQL and the low-level, procedural style of MapReduce. In 2007, it was open sourced via Apache incubator; in September 2008 it graduated to become a subproject and in 2010 Pig was accepted as a top level Apache Project. Its language is called Pig Latin [9], its shell is called Grunt and the shared repository is called PiggyBank. Penny is a framework for creating custom Pig monitoring and debugging tools. It comes with a library of about a dozen tools e.g. crash investigation, record tracing, integrity, and overhead profiling etc. Piglet is a DSL for writing Pig Latin scripts in Ruby and provides features like loops and control of flow not present in Pig [10]. A number of books and online resources are available for studying Pig [10]-[13].

IV. WORKING IN PIG

A. Installation and Execution:

Till date, a number of releases are there, latest being pig 0.10.0 which works with Hadoop 0.20.X, 1.0.X and 0.23.X. Pig runs from the machine where the Hadoop jobs are fired and need not be installed separately. However on nodes that are not a part of a cluster, it needs to be installed by following the traditional download, unpack, install and setting the classpath environment variable steps [14].

$tar –xzvf pig-0.10.0.tar.gz
$cd pig-0.10.0
$export PATH=$PATH:pig-0.10.0/bin/pig

(path setting can also be done in .bash_profile or .bashrc or /etc/profile files)

$pig –x local program.pig (or)
$pig
grunt>exec program.pig; (program.pig is a batch script)

To run pig on hadoop cluster (hadoop cluster setup [13]), set PIG_CLASSPATH environment variable to the directory containing hadoop-site.xml file.

$export PIG_CLASSPATH=/hadoop/conf
$export PATH=$PATH:${PIG_CLASSPATH}/bin
$start.sh hadoop
$pig
grunt>

Grunt is the interactive shell that permits to access HDFS and execute pig commands. Apart from the exec command, on the Grunt shell, the run command is also used for running the pig scripts and kill command like in Unix can be used for killing the MapReduce job by giving its process id.

The Pig scripts can be written using text editors or IDEs (Integrated Development Environment) e.g. Eclipse (open source software) which provide syntax highlighting and other helpful features. The syntax of a pig script can be checked without actually executing the script by using the –c, –check or –dryrun option at the command line.

At the end of script execution a summary is also displayed which shows the details of the job, the start and end time of job execution, details of Map and Reduce jobs, the input and output, some counters and an allocation graph for the dataflow between MapReduce jobs.

B. Pig Latin:

Like other dataflow languages Pig Latin also has its set of rules and syntaxes [11]-[13]. The identifier includes variable name, function name, field name, relation name etc. and starts with any letter followed by any number of letters, digits and underscores. A mixed type of case sensitivity is offered. The keywords are not case sensitive but the identifiers and user defined function names are case sensitive.

a. Data Types:

Pig supports scalar and complex data types. The scalar data types viz., int, long, float, double, chararrary and bytearray are the same as in Java programming language. e.g. seq_id:chararry, seq_length:in , gc_content:float etc.

The complex data types can contain data of any type including the complex type itself. Following are the complex data types provided in Pig-

a) ***Data map –*** A map from a string literal key to the data value(s) of any data type in the format ['key'# <value1, value2, …>]. The values in map can be of different data types. Also,the type of values can be changed by casting operation as in Java. e.g. ['seq_id'#'AC_001', 'seq_length'#23000,'percent_gc_content'#34.45]

b) ***Tuple –*** A fixed length ordered collection of data elements similar to a row in SQL. The fields can be referred by their order but need not have a schema or type associated with it. e.g. ('AC_001', 23000, 34.45) is a tuple constant with 3 fields.

c) ***Bag –*** An unordered collection of tuples that need not have a schema. All the tuples are used for describing the bag schema. It has no memory limitations except the local disk. If the bag still grows larger it is said to spill on other disks. e.g. {(‘AC_001’, 23000, 34.45) (‘AC_002’, 13040, 23.33) (‘AC_003’,6700,27.56)} is a bag with 3 tuples with 3 fields each.

b. Input and Output:

Each command of Pig Latin language produces an output typically known as a relation. Since Pig Latin is a dataflow language, it requires input to process and provides output. The output from each processing step can be fed to subsequent steps. If their names are same, they get overwritten. The Load command brings the input from the HDFS, the Store command sends the data to be stored on HDFS and the Dump command is used to view the relation on screen.

By default PigStorage is used and the data is considered to be tab-separated. Another type of storage e.g. Hbase Storage and comma-separated files can also be read by declaring specifically. e.g.

A=load ‘seq_file’ from HbaseStorage();

B=load ‘annotations’ from PigStorage(‘,’);

C=load ‘pdb_summary’ from PigStorage(id, title, date , resolution, structure_mweight);

Dump C;

Store A into /virus/genome/prot_info;

The Pig Latin statements are not executed actually until an output in the form of dump or store is requested.

c. Relational Operators:

Relational operators are applied on the input data and the output is obtained accordingly. Along with the SQL like clauses, some advanced operators are also provided in Pig Latin as discussed below:

a) FOREACH:

It is like the projection operator in relational algebra and the ‘select’ clause in SQL with field names specified but provides more functionality. It can be used to select fields, fields with arithmetic or string operators applied, producing new bags, user defined functions (UDF) etc.

A = load ‘protein’ as (prot_id, gene_id, name, start, stop);

B = foreach A generate prot_id, stop – start;

B2 = foreach A generate prot_id, $4 - $3;

C = foreach A generate ..name; -- prot_id, gene_id, name

H = foreach E generate serial, flatten(mybag) as mybag;

Flatten operator is used with foreach to remove the nesting of data in tuples and bags. Nested foreach statements are used to get the result like ‘select distinct(field)’ clause.

b) FILTER:

It works like the ‘where’ clause in SQL to select the records based on a criteria. A field is specified with some comparison operator. All the records with the matching criteria are selected. While all the operators are applicable on scalar data, only equality operators(== and !=) are applicable to tuples and maps having the same schema, equality operators are not applicable on bags.

A = load ‘pdb’ as (pdbid, title, experiment, resolution);

B = filter A by experiment matches ‘X-RAY.’;*

C = filter B by resolution <= 1; --high resolution x-ray crystal structures.

c) GROUP:

It is used to collect together the records having the same key value like SQL ‘group by’ clause but an aggregate function in the same statement is not required, rather the same key valued records are collected together in a bag.

A = load ‘pdb’ as (pdbid, title, experiment, resolution);

B = group A by experiment;

Store B into ‘pdb-exp’;

C = foreach B generate group, COUNT(experiment);

Dump C;

A Cogroup operator is also provided that works like the group operator but combines the records having similar keys from more than one input. The output is a record with a key and one bag for each input containing the grouped values from the input.

d) ORDER:

It is used to sort the records in ascending or descending order either numerically or lexically depending upon the data type of the selected field in the same way like ‘order by’ clause in SQL. Sorting is not possible for tuples, maps and bags.

A = load ‘pdb’ as (pdbid, title, release_date);

B = order A by release_date desc; -- latest releases first

e) JOIN:

It joins records from two tables based on the same field. Outer join (left or right) is also possible.

A = load ‘pdb’ as (pdbid, title, release_date);

B = load ‘reference’ as (pdbid, author, journal, title, volume, year, pubmed_id);

J = join A by pdbid, B by pdbid;

f) UNION:

It is a powerful feature of Pig that allows tables to union or concatenate even if they have different schemas. Null value is inserted into blank or unspecified fields. If both inputs have the same schema, the union also has the same schema. If the inputs are different then the schema of union is said to be unknown. Adding ‘onschema’ clause makes sure that all inputs have matching or sharable schema otherwise union is not done.

A = load ‘pdf_ref’ as (pdbid, author, journal, title, volume, year, pubmed_id);

B = load ‘pdb_not_pub’ as (pdbid, author);

C = union A, B;

g) CROSS:

The cross operator combines every record from one input with every record from another table just like the cross operator used in Set theory. This operation is used when Theta join is required, or the filter criterion is other than equi-join. The cross operation produces a lot of data; for inputs with x and y number of records, the output has x*y records, hence should be used when really needed.

C = cross A, B;

h) SPLIT:

The split operator splits the input table into a number of tables based on some condition. The records may be placed in more than two tables.

A = load 'pdb' as (pdbid, title, experiment, resolution);

Split A into H if resolution <=1, M if (experiment matches 'SOLUTION NMR');

The Pig interpreter verifies the input files and references, but the command execution takes place later when the output is required. It is known as lazy execution. It permits in-memory pipelining and query optimization. This feature is different from SQL. The table II shows some commonly used SQL statements for major database operations and its equivalent Pig Latin.

d. Advanced features:

Apart from these commonly used operators, Pig also provides a lot of advanced operators like- *distinct*: returns unique rows; *limit*: returns the specified number of rows; *sample*: returns the specified percentage of rows; *stream*: sends data to external script, etc.

A number of diagnostic operators are also available like- *describe*: shows the schema; *explain*: shows in the form of a text graph, how the script is compiled into a Map Reduce job; *illustrate*, which extracts a good sample of data using intelligent selection containing important keys and shows how the script runs; p*arallel*: causes the job to run on the specified number of reducers etc.

Table II. Major database operations in SQL and Pig Latin

OPERATION	SQL	PIGLATIN
Import CSV file into table	LOAD DATA INFILE 'data.csv' INTO TABLE table FIELDS TERMINATED BY ',';	A= LOAD 'data.csv' USING PigStorage(',') AS (field list); (data remains in file, after query operations results have to be saved again)
Copy table as another table	CREATE TABLE table2 AS SELECT * FROM table1;	A= LOAD 'table1' as (fieldlist); STORE A into 'table2' USING PigStorage(','); /-- stored as new file
Create/drop/alter view	Same as for table	Operations for view not supported.
Drop table	DROP TABLE table;	File has to be deleted using HDFS commands. $hadoop fs -rm table.txt
View schema of relation	DESCRIBE table;	DESCRIBE table;

View entire table	SELECT * FROM table;	A = LOAD 'table' AS (field list);
		B = FOREACH A GENERATE *;
		DUMP B;

View columns by name	SELECT col1, col2 FROM table;	B = FOREACH A GENERATE col1, col2;
		DUMP B;

Sort in descending order	SELECT * FROM table ORDER BY col1 DESC;	B = ORDER A BY col1 DESC;
		DUMP B;

Joining two tables on same column values	SELECT * FROM table1, table2 WHERE table1.col=table2.col;	B = LOAD 'table2' AS (field list);
		C = JOIN A ON col, B ON col;
		DUMP C;

Cross product of two tables	SELECT * FROM table1, table2;	C = CROSS A, B;
		DUMP C;

Count of rows in a table	SELECT COUNT(*) FROM table;	B = GROUP A all;
		C = FOREACH B GENERATE COUNT(A);
		DUMP A;

Count of distinct values in a column of table	SELECT col, COUNT(DISTINCT(col)) FROM table;	B = FOREACH A GENERATE col;
		B = DISTINCT B;
		C = GROUP B BY col;
		D = FOREACH C GENERATE group as col, COUNT(B);
		DUMP D;

View records based on condition	SELECT * FROM table WHERE col operator value ;	B = FILTER A BY fieldname operator value;
		DUMP B;

V. RESULTS AND DISCUSSION

The RCSB Protein Data Bank that archives information about the experimentally determined structures of proteins, nucleic acids and assemblies; which is available online at http://www.rcsb.org/pdb/home/home.do. A report of the structures available (as on July 24, 2012) is prepared that contains the PDB id, structure title, experimental method and resolution for 83266 structures and named 'pdb.txt'. A similar report is prepared for 4959 structures derived from viral organisms and named 'virus_pdb.txt'. Pig Latin commands are used to group the structures together based on the experimental method and the results are reported.

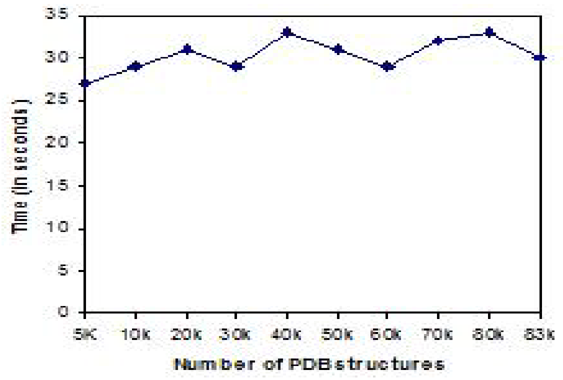

Figure 1. Graph between number of structures and time taken to process them completely.

The script is run on a single cluster RHEL5 machine with two Intel® quad core Xeon processors and IBM® BigInsights framework installed.

The result for virus structure set was obtained in 27 seconds and result for entire pdb set was obtained in 30 seconds as reported from the BigInsights Administrator console. Hadoop is said to be more efficient with large sized files. Fig. 1 shows a plot between different number of structures taken and time taken to process the entire set of queries.

Following are the commands issued and the output obtained while processing both the text files containing structure information. "hadoop fs -put" command is a hadoop file system command that loads the file into HDFS and the "pig" command launches the grunt shell.

```
$ hadoop fs -put /download/virus_pdb.txt virus_pdb.txt
$ pig
grunt> A = load 'pdb.txt' as (id:chararray, exp:chararray, date:chararray, res:float);
grunt> B = group A by exp;
grunt> C = foreach B generate group, COUNT(A);
grunt> dump C;
INFO [Thread-20]
org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
INFO [main] org. apache. hadoop. mapreduce. lib. input. File Input Format - Total input paths to process : 1
(SOLUTION NMR,9467)
(SOLID-STATE NMR,52)
(EPR, SOLUTION NMR,1)
(FIBER DIFFRACTION,37)
(X-RAY DIFFRACTION,73093)
(POWDER DIFFRACTION,18)
(ELECTRON MICROSCOPY,441)
(NEUTRON DIFFRACTION,38)
(SOLUTION SCATTERING,32)
(FLUORESCENCE TRANSFER,1)
(INFRARED SPECTROSCOPY,4)
(EPR, X-RAY DIFFRACTION,5)
(ELECTRON CRYSTALLOGRAPHY,33)
(SOLID-STATE NMR, SOLUTION NMR,3)
(SOLUTION NMR, X-RAY DIFFRACTION,1)
(THEORETICAL MODEL, SOLUTION NMR,7)
(SOLUTION NMR, NEUTRON DIFFRACTION,1)
(SOLUTION NMR, SOLUTION SCATTERING,3)
(SOLUTION SCATTERING, SOLUTION NMR,8)
(SOLID-STATE NMR, X-RAY DIFFRACTION,1)
(NEUTRON DIFFRACTION, X-RAY DIFFRACTION,8)
(X-RAY DIFFRACTION, NEUTRON DIFFRACTION,10)
(SOLUTION SCATTERING, ELECTRON MICROSCOPY,2)
```

A similar script is used to determine the number of structures having different resolutions using the split command.

```
grunt>split A into P if resolution <=1, Q if (resolution>1 and resolution <=2), R if (resolution>2 and resolution<=3), S if resolution>3;
```

The number of records in all the relations as obtained above are counted as:

```
grunt>X = group P all;
grunt>Y = foreach X generate COUNT(P);
grunt>dump P;
```

The count of X-ray crystal structures grouped according to the resolution range is provided on the PDB homepage. For the above used data set (pdb.txt), the count given on the site (as on July 24,2012) and a similar result obtained by Pig Latin scripts is given in table III.

Table III. Count of X-ray crystal structures by resolution.

As reported on PDB site	Output of Pig script
Less than 1.5 Å : 5343	<= 1Å : 495
>=1.5 Å and < 2.0 Å : 24364	>1 Å and <=2Å : 34561
>= 2.0 Å and < 2.5 Å : 24910	>2 Å and <=3Å : 34224
>= 2.5 Å and < 3.0 Å : 13196	>3 Å : 4103
3.0 Å and more : 5304	

VI. CONCLUSION

The new high level language Pig and its query language Pig Latin can be successfully used for large data set analyses. Using the relational algebra and SQL approach, complex queries can be designed in a very efficient manner, as it is broken into a number of simple steps. At each step the output can be seen using dump command and refinements can be done. Also there is no need to maintain large database tables in DBMS as it can directly process from the files in HDFS. Large sequence processing tasks, similarity matching, and alignments are the types of jobs, which will be efficiently handled by Pig. After an initial test run on a small data set, big runs can be applied to large files (in GBs/TBs) on large clusters. Although there are some overheads for compiling commands into MapReduce jobs and lazy execution features seem to slow down small queries; still it provides the flexibility and advantages of using Hadoop as underlying platform like fault tolerance, efficiently handling large sized files, load balancing etc.

VII. ACKNOWLEDGMENT

This work was supported in part by the University of Pune and the Centre of Excellence grant to Bioinformatics Centre by Ministry of Communications and IT (MCIT), and Department of Biotechnology (DBT), Government of India.

VIII. REFERENCES

[1] http://en.wikipedia.org/wiki/Bioinformatics

[2] http://en.wikipedia.org/wiki/Next-generation_sequencin g

[3] http://en.wikipedia.org/wiki/Apache_Hadoop

[4] Ronald C Taylor, "An overview of the Hadoop/MapReduce/Hbase framework and its current applications in bioinformatics," BMC Bioinformatics, 11(Suppl 12):S1, 2010.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters", in Proc. 6th OSDI, Dec. 2004, pp. 137-149.

[6] S. Ghemawat, H. Gobioff, S. T. Leung, “The Google file system”, in Proc. 19th ACM Symposium on Operating Systems Principles, 2003, Lake George, NY, ACM Press, pp. 29-43.

[7] J. Dean and S. Ghemawat, “MapReduce: A Flexible Data Processing Tool”, Communications of the ACM 2010, 53(1), pp. 72-77.

[8] Google blesses Hadoop with MapReduce patent license. [http://www.theregister.co.uk/2010/04/27/google_licenses_mapreduce_patent_to_hadoop/].

[9] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, “Pig Latin: A Not-So-Foreign Language for data Processing”, In Proc. SIGMOD’08, 2008, Vancouver, Canada.

[10] https://cwiki.apache.org/confluence/display/PIG/PigTools

[11] Alan Gates, Programming Pig, O’Reilly Media, 2011.

[12] Tom White, Hadoop: The definitive Guide, O’Reilly Media, Third edition May 2012.

[13] http://pig.apache.org/docs/r0.10.0/index.html

[14] http://hadoop.apache.org/common/docs/r0.20.2/quickstart.html