Research of a MapReduce Communication Data Stream Processing Model

Wenchuan Yang
School of Computer, Beijing University of Posts and Telecommunication, Beijing,China,100876
Century College, BUPT, Beijing,China,102613
e-mail: yangwenchuan@bupt.edu.cn

Bei Jia
Xi'an Communication Institute,
Shaanxi, Xi'an, China, 710106
e-mail: jiabeihn@gmail.com

***Abstract*—In this paper, we propose CDS-MR , a MapReduce deep service analysis system based on Hive/Hadoop frameworks. Normally, the job of the switch is to transmit data. There is a tendency to put more capability into the switch, such as retain or query pass by data. Thus we definitely need to think about what can be kept in working storage and how to analysis it. Obviously, the ordinary database cannot handle the massive dataset and complex ad-hoc query. MapReduce is a popular and widely used fine grain parallel runtime, which is developed for high performance processing of large scale dataset.**

Keywords-Cloud Computing; Hadoop; Hive; MapReduce; Workflow

I. INTRODUCTION

Many distributed computing models have been developed for high performance processing of large scale scientific data. MapReduce is a popular parallel programming model proposed by Google to support large-scale data processing. Hadoop is an open source implementation of MapReduce with a distributed file system (HDFS) [1,2].

Each MapReduce job takes a set of key/value pairs as input, and produces a set of key/value pairs. The computation of MapReduce jobs is split into 2 phases: map and reduce[3].

In map phase, map function takes an input key/value pair, read the data accordingly from HDFS, and produces a set of intermediate key/value pairs[4]. In reduce phase, each reduce operation accepts an intermediate key and all values associate with that key. It merges these values to form a possibly smaller set of values, emits key/value pairs of the final output, and writes the final output to HDFS[5].

Hive in CDS-MR provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL[6]. It has traditional SQL constructs like joins, group bys, where, select, from clauses, and from clause subqueries.

It tries to convert SQL commands into a set of MapReduce jobs. Apart from the normal SQL clauses, it has a bunch of other extensions, like the ability to specify custom mapper and reducer scripts in the query itself, the ability to insert into multiple tables, partitions, HDFS, or local files while doing a single scan of the data.

II. SYSTEM ANALYSIS

A. Switch and Its Data

The current job of the switch in CDR is to transmit data and not to retain it or query it. But for deep service analysis, we need to copy, retain and analysis the stream data passed by the switch.

These streams have the following abilities.

Increased rapidly, big data: Each produces 700M every 5 minutes, and daily up to 1.5 terabytes arriving for each province. Almost 1 petabyte for 31 province each week.

Write once, read many times: The deep stream analysis is focused on the global feature and mainly focused on batch access for decision making. Seldom update and insert operations in it.

Dynamic schema, low integrity: Since different service streams enter the system. Each stream can provide elements at its own schedule; they need not have the same data rates or data types, and the time between elements of one stream need not be uniform.

Not fit for RDBMS: The fact that the rate of arrival of stream elements is not under the control of the system distinguishes stream processing from the processing of data that goes on within a database-management system.

Thus we definitely need to think about how and what can be kept in working storage and how to analysis them.

B. Data Analysis

Applications often require more resources than are available on an inexpensive machine. The need for efficient and effective models of parallel computing is clear due to the existence of extremely large datasets which cannot be processed without using multiple computers.

There are two interrelated but distinct classes of uses for analyzing data:

1)Producing daily and hourly summaries over large amounts of data. These summaries are used for a number of different purposes within the company. Reports based on these summaries are used by engineering and non-engineering functional teams to drive product decisions. These summaries include reports such as amount of send/receive packages for specific applications.

2)Running ad-hoc jobs over CDR data. These analyses help answer questions from 3G App providers and executive team. It could be examined only under special circumstances using time-consuming retrieval processes. There are also lots

of Hive tables, into which summaries or parts of streams may be placed, and which can be used for answering queries.

Both the statistic and ad-hoc queries can rely on sampling data, depending on how fast we need to process queries. Sometimes it is enough to get the decision information just based on the sampling data. Usually it is not necessary to store all the stream data as for sufficiently limited capacity.

III. PROTOTYPE DESIGN

A. Data Architecture

The basic components of our architecture and the data flow within these components are shown in Figure 1.

The following components are used in processing data:

HDFS

A large fraction of this CDR data is copied into one central HDFS instance. Data from the switching stream data is continuously replicated to the HDFS cluster by copier jobs. The CDR devices are mounted on the Hadoop tier and the copier processes run as map-only jobs on the Hadoop cluster. This makes it easy to scale the copier processes and makes them fault-resilient. Currently, we copy over 1.5 TB per day from CDR to HDFS in this manner.

Hive/Hadoop

We use Hive to build a data warehouse over all the data collected in HDFS. Files in HDFS, including user data from CDR and statistic data from the Hive, are made available as tables with logical partitions. A SQL-like query language provided by Hive is used in conjunction with MapReduce to create/publish a variety of summaries and reports, as well as to perform historical analysis over these tables.

Tools

Browser-based interfaces built on top of Hive allow users to compose and launch Hive queries (which in turn launch MapReduce jobs) using just a few mouse clicks

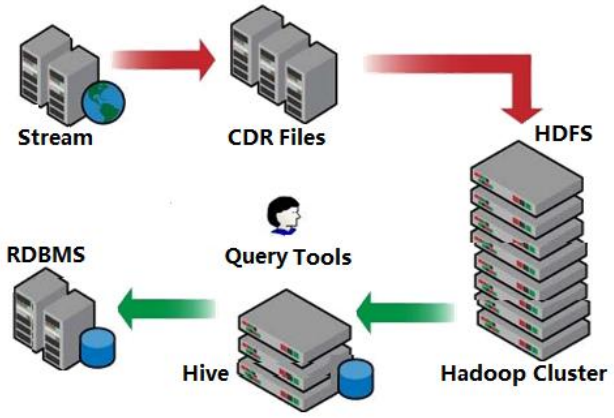

Figure 1 Architecture of CDR processing

B. Hadoop Cluster

Hadoop cluster adopts master-slave architecture. The master node runs a namenode, a secondary namenode and a job tracker. The namenode is mainly used for HDFS for hosting the file system index. The secondary namenode can generate snapshots of the namenode's memory structures, thus preventing file system corruption and reducing loss of data; the job tracker allocates work to the task tracker nearest to the data with an available slot.

The slave node runs a datanode and a task tracker: datanode contains blocks of data inside HDFS where multiple datanodes can serve up distributed data over network; task tracker will spawn java processes as workers to execute the work received from job tracker. For reliability, file data is replicated to multiple storage nodes[7].

There are two types of nodes that control the job execution process: a jobtracker and a number of tasktrackers. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers. Tasktrackers run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job[8]. If a task fails, the jobtracker can reschedule it on a different tasktracker[9].

Hadoop divides the input to a MapReduce job into fixed-size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the user-defined map function for each record in the split.

Having many splits means the time taken to process each split is small compared to the time to process the whole input.

The HDFS that underlies MapReduce provides efficient and reliable distributed data storage required for applications involving large datasets.

The basic function of the MapReduce model is to iterate over the input, compute key/value pairs from each part of the input, group all intermediate values by key, then iterate over the resulting groups and finally reduce each group. The model efficiently supports parallelism.

C. Analysis over Hive

Hive is a data warehouse infrastructure built on top of Hadoop and serves as the predominant tool that is used to query the data stored in Hadoop at CDS-MR.

The natural consequence of these requirements was a system that could model data as tables and partitions and that could also provide a SQL-like language for query and analysis. Also essential was the ability to plug in customized MapReduce programs written in the programming language of the user's choice into the query.

Additionally, the ability provided by Hive in terms of expressing data pipelines in SQL can and has provided the much needed flexibility in putting these pipelines together in an easy and expedient manner.

This is especially useful for organizations and products that are still evolving and growing. Many of the operations needed in processing data pipelines are the well-understood SQL operations like join, group by, and distinct aggregations. With Hive's ability to convert SQL into a series of Hadoop MapReduce jobs, it becomes fairly easy to create and maintain these pipelines.

In one of the optimizations that is being added to Hive, the query can be converted into a sequence of Hadoop MapReduce jobs that are able to scale with data skew. Essentially, the join is converted into one MapReduce job .

IV. EXPERIMENT

To address the challenges faced in designing and setup large MapReduce, we have implemented a MapReduce prototype and do some fundamental experiment.

The sentence chunking application consists of five steps: 1) Setting up a Hadoop Cluster, 2) Importing the CDR file to HDFS, 3) Constructing a MapReduce Model, 4) Running the processing and get Results, 5)Providing a statistic and Ad-hoc query tool for user.

We will discuss these steps and Query tool as below.

A. *Hadoop Cluster*

Firstly, we will explain how to set up Hadoop to run on a cluster of machines. Our experiment Hadoop environment is designed to run on commodity hardware. That means that you are not tied to expensive, proprietary offerings from a single vendor; rather, you can choose standardized, commonly available hardware from any of a large range of vendors to build your cluster.

We have 1 PC server as Namenode and 20 PC as data node to construct commodity cluster. The salient point is that the aggregate band-width between nodes on the same sub-switch is much greater than that between nodes on different switch. While the hardware specification for your cluster will assuredly be different, Hadoop is designed to use multiple cores and disks, so it will be able to take full advantage of more powerful hardware

In this case, we will write a program that mines CDR data. The 1.5 terabyte data is produced by a provincial CDS data center, which is a good candidate for analysis with MapReduce, since it is semi-structured and record-oriented.

The data is stored using a line-oriented ASCII format, in which each line is a record. The format supports a rich set of 3G service elements, many of which are optional or with variable data lengths. For simplicity, we shall focus on the basic elements, such as Phone Number, IP Address which are always present and are of fixed width.

Data files are organized by date center number and date. There is a directory for each day, which contains a file with its readings from that center.

To visualize the detail record, consider the following sample lines of input data (some unused columns have been dropped to fit the page):

```
...QQTC201206131127201206131132101 40...117710229...
...SNTP201206131131201206131132101 40...11771027...
...HTTP201206131127201206131132101 40...117710221...
...MRDR201206131129201206131132101 41...117710217...
```

B. *MapReduce Model*

MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function.

The input to our map phase is the raw CDR data. We choose a text input format that gives us each line in the dataset as a text value. The key is the offset of the beginning of the line from the beginning of the file, but as we have no need for this, we ignore it.

Our map function is focused on pulling out the necessary parameter as: application class, start Time, source IP address, phone number, send package, receive package, etc. Since these are the fields we are interested in.

In this case, the map function is just a data preparation phase, setting up the data in such a way that the reducer function can do its work on it: finding and counting the necessary data for deep service analysis. The map function is also a good place to drop bad records: here we filter out data that are missing, suspect, or erroneous.

These lines are presented to the map function as the key-value pairs:

```
(0,   QQTC201206131127... 014088182138...162518dial1...)
(128, SNTP201206131212... 028028182138...152513ip1...)
(256, HTTP201206131317... 016088182137...127521dial1...)
...
```

The keys are the line offsets within the file, which we ignore in our map function. The map function merely extracts the necessary data segment and emits them as its output. For example, the application class and send package can be interpreted as:

```
(QQTC, 21)
(SNTP, 17)
(HTTP, 11)
...
```

The output from the map function is processed by the MapReduce framework before being sent to the reduce function. This processing sorts and groups the key-value pairs by key. So, continuing the example, our reduce function sees the following input:

```
(QQTC, [21, 16 ,22...])
(SNTP, [17, 10...])
(HTTP,[11, 6...])
```

Each application class with a list of all its sending package. All the reduce function has to do now is iterate through the list and pick up the sum for each reading:

```
(QQTC, 4621)
(SNTP, 1022)
(HTTP, 502)
```

This is the final output: the total send package in each application class at a time

V. STATISTIC AND QUERY

A. *Data analysis*

Usually 3G carrier want to know statistic data such as which application is mostly used by the smart phone user at a time, daily or hourly reports for send and/or receive packages for specific app class and/or IP address . Since the bulk of the data(because the total data for a popular 3G dataset is huge), analysis tasks cannot be performed by RDBMS.

Fortunately, the stream datasets in HDFS are structured and could be easily partitioned. Hive and Hadoop can be easily used for such data analysis and statistic applications.

Our experiment is a typical application in the online CDR data analysis. All of these are easily expressible in Hive using a couple of SQL queries (that would, in turn, generate multiple Hadoop jobs).

Here different data, such as application class and send and/or receive packages, can be structured as tables in Hive. Following is the detail steps.

(1). Create Hive table;

(2). Load data

(3).Group by application class for send packages.

(4). List Result as,

...

Hive is proved as easily useful for assembling statistic data and then feeding the same into a data analysis engine.

From this statistic info in Hive table, the daily or hourly reports for send and receive package can be produced according to application class and IP address, etc.

If only an estimate were required, the same queries can be run for a sample set of the users using sampling functionality natively supported by Hive.

Then, the standard statistic info can also be deduced under this framework. This would involve sampling CDR data, grouping it by time and then finding the number of data model at different time points via a custom reduce script.

B. Ad-hoc analysis and product feedback

Although Hive and Hadoop are batch processing systems that cannot serve the computed data with the same latency as a usual RDBMS, they are to be able to support ad-hoc analysis and product feedback solutions.

For example, some 3G Application(e.g. QQ) provider usually makes product changes, and it is typical for product managers to understand the impact of a new feature, based on user engagement as well as on the user behavior.

The product team may even wish to do a deeper analysis on what is the impact of the change based on gender and age. A lot of this type of analysis could be done with Hadoop by using Hive and regular SQL.

The measurement of usage can be easily expressed as a join of the user information(age or/and gender) for the particular related feature in CDR. It can be used to compute the effect of 3G app changes on different age or/and gender.

In our experiment, the QQ provider need to do ad-hoc query for packages sent by male at a time. A HiveQL OuterJoin can be used to do this as shown in Figure 2. Assume the CDR_Table provides the CDR data, and User_Table contains gender info as User_Table (PhoneNo string , Name string , gender tinyint).

Data in User_Table is listed as,

13800127XXX Tom 1
13701166XXX Alice 0
13911245XXX John 1
...

Some of this analysis needs the use of custom map and reduce scripts in conjunction with the Hive SQL, and that is also easy to plug into a Hive query.

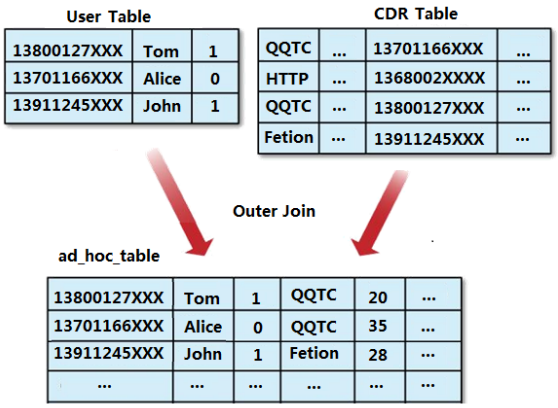

Figure 2 Outer Join for ad-hoc table

VI. CONCLUSIONS

In this paper we proposed and implemented a new distributed workflow system CDS-MR. It is a MapReduce systems based on composition of Hadoop and Hive. The architecture of CDS-MR is described. CDS-MR is extended to support both statistic and ad-hoc queries. In our experiments, we demonstrated that the CDS-MR is fit for deep service analysis, which explains the motivation for our CDS-MR.

ACKNOWLEDGMENT

This paper is supported by the Opening Project of State Key Laboratory of Digital Publishing Technology.

REFERENCES

[1] K. Arrow. Aspects of the theory of risk-bearing. Helsinki: Yrjo Jahnsson Lectures, 1965.

[2] A. AuYoung, L. Grit, J. Wiener, and J. Wilkes. Service contracts and aggregate utility functions. In Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC), June 2006.

[3] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In ACM SIGMOD: International Conference on Management of Data, 2007.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A.Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In Proceedings of the ACM Symposium on Operating Systems Principles, 2003.

[5] R. E. Bryant. Data-intensive supercomputing: The case for DISC. Technical Report CMU-CS-07-128, Carnegie Mellon University, 2007

[6] K. Cardona, J. Secretan, M. Georgiopoulos, and G.Anagnostopoulos. A grid based system for data mining using MapReduce. Technical Report TR-2007-02, AMALTHEA, 2007.

[7] B. N. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat. Mirage: A microeconomic resource allocation system for SensorNet testbeds. In Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors, 2005.

[8] B. N. Chun and D. E. Culler. Market-based proportional resource sharing for clusters. Technical Report CSD-1092, University of California at Berkeley, Computer Science Division, January 2000.

[9] B. N. Chun and D. E. Culler. User-centric performance analysis of market-based cluster batch schedulers. In Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid, 2002.