FIND
A Data Cloud Platform for Financial Data Services

Zhicheng Liao, Yun Xiong and Yangyong Zhu
Research Center for Dataology and DataScience, School of Computer Science, Fudan University,
825 Zhangheng Road, Shanghai, China

Keywords: Cloud Computing, Data Cloud, Securities Data Integration, Data Service.

Abstract: In recent years, researchers have paid more interest in dealing with large scale of data. However, it is difficult to discover patterns from various sources of big data flexibly and efficiently. In this paper, we design a data cloud platform for financial data services (FIND), and implement a prototype system to evaluate the performance and usability of the data cloud. FIND consists of a cloud infrastructure, a data resource center and a data service portal. FIND provides high performance computation capability, high quality integrated financial data, sophisticated data mining algorithms, and powerful data services.

1 INTRODUCTION

The growth of data is much faster than we can imagine. Data can be broadly classified into three types: structured data, unstructured data and semi-structured data (Buneman, 1997). For most centralized information system, a user may face to thousands of data and the manual operation is almost impossible when he want to get different data from different systems and merges them together. There are several approaches to process the big data. Using a cluster or a distributed system is a better way. By constructing grid computing and cloud computing infrastructures in commercial systems, researchers have found a low-price and high-scalable way to deal with big data. No matter what method you choose, it cannot reduce the workloads when users want to discover some useful information from large amount of data. Some data providers use a data warehouse and an ETL tool to integrate multiple sources of data. Data mining tool is also used to give a better result to user's query. If all data is structured data, it is a good solution. However, if integrating unstructured data and/or semi-structured data, it may cause loss of some information in converting them to structured data.

As discussed above, existed solutions focus on either processing big data or integrating various sources of data. In financial industry, the data is extremely huge and variety. Many financial models are too complex to run with the big data on an ordinary server. For most users, the data is too expensive to get a complete copy, and they also have no capability to process those big data. Therefore, we need to provide data services for users instead of providing data. Users only need to pay for services they used. We need a financial data service platform to support processing large amount of financial data, integrating various types of data, and providing financial data services. Our contributions include:

- We design a data cloud platform for financial data services (FIND). It is based on cloud computing technique. It has capability of integrating huge amount of hybrid data. FIND provides high quality integrated financial data and public financial data services, and users pay for using services on demand instead of owning data.
- We design several data mining algorithms in FIND, and build a data mining algorithm library for user to use the data easily and effectively.
- We implement a prototype system to evaluate the performance and usability of FIND by experiments. Experiments show FIND has good performance in processing big data, integrating various data, and providing data services.

The rest of the paper is organized as follows. Section 2 is related work. In section 3, we introduce the design of a data cloud platform FIND. In section 4, we show a prototype system building with Apache Hadoop and HBase, and carry out some experiments to prove the usability of FIND.

Liao Z., Xiong Y. and Zhu Y..
FIND - A Data Cloud Platform for Financial Data Services.
DOI: 10.5220/0003990701170122
In *Proceedings of the International Conference on Data Technologies and Applications* (DATA-2012), pages 117-122
ISBN: 978-989-8565-18-1

2 RELATED WORK

To manage various types of data in a single database management system and a uniform data schema, Google introduced a new simple data model database management system named Bigtable (Chang, et al., 2006). It is a distributed scalable non-relational database. open source projects that implement Bigtable, such as Apache HBase, Zvents HyperTable, etc.

Using a super-computer can process large volumes of data, but the cost is extremely high. So some researchers are trying to use distributed computing techniques to provide as high computation capability, but in a lower cost. Google first introduced the term "Cloud Computing" in 2006, and gives a chance to process big data in a low cost. The Apache Hadoop project is a famous open source software for cloud computing. Amazon Web Services (AWS) and Google App Engine are typical commercial cloud computing services.

Some researchers are trying to find new programming paradigms to process vary large datasets, especially to solve complex problem in machine learning, bioinformatics, mathematical finance, and so on. Dean and Ghemawat (2004) introduced a novel programming framework named MapReduce (Dean and Ghemawat, 2004) which supports distributed computing on large datasets on clusters of computers. Ghoting et al. (2011) present the SystemML (Ghoting et al., 2011), which machine learning algorithms are expressed in a high-level script language, and subsequently compiled and executed in Hadoop. Cordeiro, Traina, Junior, Traina, López, Kang and Faloutsos (2011) give a new way to use MapReduce for clustering over a very large moderate-to-high dimensionality dataset (Cordeiro, et al., 2011). Chen, Wang and Zhu (2006) had built a Metropolis Shared Research Platform (SRP) with grid computing infrastructure (Chen, et al., 2006) to integrate the scientific data, archives and other objects. Sul and Tovchigrechko (2011) give a MapReduce version of BLAST algorithm used in bioinformatics, (Sul et al., 2011). Vecchiola, Abedini, Kirley, Chu and Buyya (2010) present and implement a co-evolutionary learning classifier system on public cloud Amazon EC2 (Vecchiola et al., 2010).

These researches focus on either processing big data or integrating single domain-specific data. In practice, users always hope the system to afford data as much as possible, provide services as flexible as possible, and give the results as simple but useful as possible. Therefore, it is necessary to design and implement a system that can integrate various types of data, process the big data, and provide diversified data services. Aiming at providing financial data services, we need to design a powerful platform that has capability of high performance computing and providing elastic services.

3 ARCHITECTURE DESIGN

Now we present a platform for financial data services, named FIND, which collects and integrates financial data from multiple sources, and provide various public services, such as querying, searching, statistical analyzing, data mining, etc.

3.1 Architecture

Considering the capability of processing big data and hybrid data, we choose the cloud computing technique as the base of FIND. We logically divide FIND into three parts: a *cloud infrastructure*, a *data resource center* and a *data service portal*. The cloud infrastructure gives basic hardware support, such as host, network device, storage device, etc., and provides some interfaces to access and manage the underlying hardware. The data resource center holds all of data in FIND, i.e. the original data, metadata, ontologies, and so on. The data service portal faces to end users, and provides public services of FIND. The architecture of FIND shows as Figure 1.

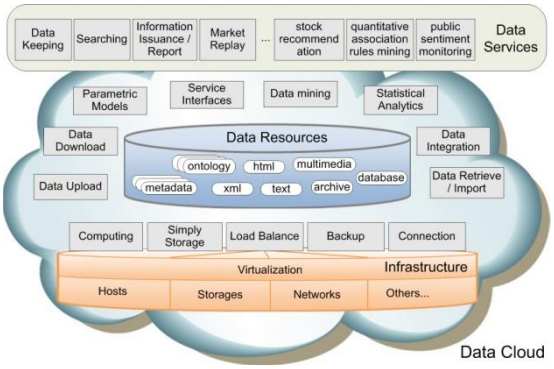

Figure 1: The FIND architecture.

The cloud infrastructure and the data resource center virtually have four internal service layers shown as follows.

Infrastructure Service Layer (ISL): to provide hardware support and system software support to other internal and public services.

Data Service Layer (DSL): to provide data maintaining and simple statistical analysing such as

summing, counting, averaging. Other programs store or retrieve the data via certain interfaces.

Platform Service Layer (PSL): to provide common programming routines, data mining algorithm libraries, standard business process procedures, parametric models, and so on.

Software Service Layer (SSL): to provide well-defined and easy-to-use applications for end users.

Although the four internal service layers looks like hierarchical, it does not mean one service layer must build on another service layer. They are flat. The data service portal deploys many programs that provide various types of public financial data services to end users.

3.2 Data Resource Center

The data resource center consists of a distributed file system and a distributed scalable non-relational database. It also contains some standalone relational databases on the specific hosts. The distributed scalable non-relational database holds most of data and the distributed file system stores files. We retrieve data (including files) from different sources, for example, crawling from the Web, importing from external databases, uploading or inputting by users. And these data are in several subjects, such as government notice, economy news, market/trade data, index, forum thread, and so on.

FIND integrates these data in a proper way and puts them into the distributed scalable non-relational database. For structured data come from a relational database, we re-design the schema and transform tuples by copying or mapping attributes, or simply import the schema to the distributed scalable non-relational database by using a column family as the relation and column qualifiers as attributes in that relation. The latter is easy to implement but may be difficult to use. For other structured data that are not from relational databases, we design some common column families as subjects, and append attributes of structured data to the corresponding column family.

For unstructured data, such as a text file or a multimedia clip, we save the file to the distributed file system, and save the file name, length, keywords, and other retrievable embedded attributes (if exists, such as version information or document variables) to the non-relational database. For semi-structured data, we regard it as unstructured data but extract more attributes and most of contents from it.

The data resource center also has a lot of component libraries, internal functions and interfaces that are used in other programs. It provides following internal functions:

Data Query: to query data from data resource center, by keyword or certain conditions.

Data Download: to download data that is matched the conditions.

Data Upload / Save: to support users or programs upload or save data to the data resource center.

Data Retrieve/Import: to retrieve data from other data sources, especially from internet or other file systems. It also supports importing data from an external relational database or a record file.

Statistical Analytics: to provide some basic statistical algorithms, such as summing, counting, maximum / minimum selecting, etc.

Data Integration: to integrate special sets of data (in most cases they are retrieved from different data sources), by performs data cleaning, extracting, translating, loading and other necessary steps.

Data Mining: to provide advanced data analysing and data mining algorithms libraries. It can perform more complex statistical analysing and common data mining algorithms, including pattern matching, clustering, classification, outlier detection, etc. Other programs invoke these algorithms and reach their data mining targets.

Build-in Components: include most of program libraries and components, parametric models, script interpreters, etc. Parametric model libraries provide some sophisticated parametric data processing models and business models. Programmer only needs to determine the parameter values, and then the model will give a good result.

Interfaces: include program interfaces, web service interfaces and cloud management interfaces. Program interfaces support to load, run, suspend or terminate a program. Web service interfaces support a program to communicate with other services or programs. Cloud management interfaces support the most internal managing operations of FIND, such as job dispatching, task scheduling, message queuing, intermediate data maintaining, etc.

3.3 Data Service Portal

The data service portal deploys a lot of software and provides public data services to end users. Users could use the software without any customization. Generally depending on the software execution policy, any user permits to execute software, or only an authorized user allows executing. Software implements the basic and common functions that users want to use, for example, market replaying, searching, data mining or information publishing.

FIND also provides some business models, for

example, personalized recommendation mining, quantitative association rules mining, frequent pattern mining, public sentiment mining and monitoring, and so on. These models are based on the data mining algorithms FIND provides, and using the data in the data resource center.

If necessary, FIND will provide some "pass-through" services, and allow specific users to access data resources directly. For example, data keeping service allows users to store their private data to the data resource center and access them everywhere.

The primary workflow of using these public data services is as following. A user accesses the data service portal, chooses the software she or he wants to run, and submits a software execution request. By the policy of use, the user may pay the usage fee before executing. If all of the execution conditions are met, the portal locates the software package, creates a private execution environment, loads the software into it and calls the program interface to run. The portal gathers the result and feedbacks to the user via the web service interface.

4 PROTOTYPE SYSTEM AND EXPERIMENTS

We have constructed a prototype system of FIND with 6 desktop computers, named Cloud001 to Cloud006. The main hardware of the computer is: Intel I5-650/3.2GHz CPU, 4GB RAM, 1TB hard disk drive, 1GB RJ-45 NIC. The operating system is RedHat Enterprise Linux server 5.5 x86_64-bit version. We use Hadoop, HDFS and HBase to set up FIND. The NameNode of FIND is running on Cloud001, and it is also the master node of HBase. All PCs link to a Cisco C2960G gigabits switch.

We choose two financial problems as experiment background. One is an implementation of Monte Carlo option model (Boyle, 1977) for solving option valuation problems. It will test out the high performance processing capability of FIND. The other experiment is stock index calculation. It will test out the feasibility of FIND in data integrating, service workflow and continuous computing.

4.1 Monte Carlo Option Model

We implement the Monte Carlo option model in a MapReduce program and run it on FIND. The global parameters for the model are: the initial price $S_0=100$, the exercise price of the option $E=90$, Expiration date that represents in proportion of year $T=0.1$, risk free rate per day $r=0.12/365$ and the variance rate (given in square root) $\sigma=0.1$. We also assume the probability distribution is Gaussian.

The Map task iterative calculates the option value on simulating only once per input path. The Reduce task collects the intermediate results of the Map task and gives the average value as the final result. The pseudo-code of the Map task is shown as following:

```
Map task: calculate simulation value
    Input: <path index, any value> pair
    Output: <"option", payoff>
    Global parameters: S0=100, E=90,
          T=0.1, r=0.12/365, σ=0.1

1.  Δt=1/365;
2.  days=T/Δt;
3.  st=exp((r -σ*σ/ 2)* Δt);
4.  Temp= S0;
5.  for d=0, 1, 2...days-1 do
6.      Temp=Temp*st*exp(σ*sqrt(Δt)*
                rand.nextGaussian());
7.  end for
8.  payoff=max(Temp - K, 0)
9.  payoff=payoff  >  0  ?  payoff  *
                Math.exp(- r * T) : 0;
10. return <"option", payoff>;
```

We use a constant string "option" as the key of intermediate output data of the Map task. So the Reduce task will combine all values at once. The pseudo-code of the Reduce task is shown as following:

```
Reduce Task: calculate average value
    Input:  "option"  and  corresponding
           payoff[]
    Output: average value

1.  Sum=0;
2.  Count=0;
3.  For each v in payoff[] do
4.      Sum = Sum + v;
5.      Count++;
6.  end for
7.  return Sum/Count;
```

For comparing, we run the same job on a server cluster. The cluster includes two servers. The main hardware configuration is: Intel Xeon E5640/2.66GHz*2 CPUs, 32GB RAM, 146GB*3 hard disk drives in RAID 5 mode, 1GB RJ-45 NIC*2. The operating system is RedHat Enterprise Linux server 5.5 x86_64-bit version with integrated cluster suite, and the fencing device is the motherboard integrated Intelligent Platform Management Interface (IPMI).

Table 1: The running time, price and variance of the multi-threaded program.

Number of Threads	Running time (ms)		Option price		Theoretical value	σ of price	
	1.6M	25.6M	1.6M	25.6M		1.6M	25.6M
1	100897	-	0.799412	-	0.798	0.001769	-
2	50252	807257	0.797454	0.798014	0.798	0.000684	0.000018
8	19837	264827	0.798513	0.798132	0.798	0.000643	0.000165
16	17174	228186	0.798680	0.798681	0.798	0.000852	0.000853
32	12945	215037	0.799242	0.798609	0.798	0.001556	0.000763

We run the model with 1.6M and 25.6M simulation paths input on FIND and the cluster, and each case runs 5 times. The *running time* is the time span from job initializing to job finished (accurate to millisecond) and σ is the standard variance of five corresponding prices.

Figure 2 shows the running time of each case. It is easy to discover that the FIND runs faster than cluster in both 1.6M and 25.6M paths cases. Each server in the cluster is more powerful than PC in the data cloud, but the FIND is advantageous because of the nature of the cloud architecture.

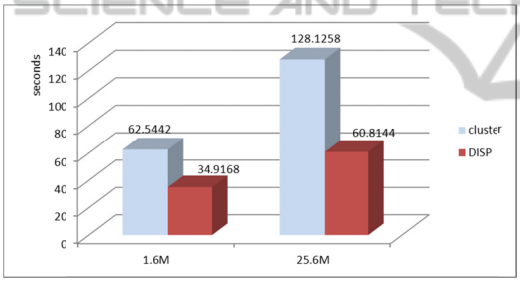

Figure 2: The running time compares on number of paths.

Figure 3 shows the variance of each result. Because the generated random number sequences are not exactly same in each time, the price would be a slightly different. We have discovered that result lines of 25.6M are smoother than 1.6M's. The acceptable reason is job of 25.6M paths are run separated in 4 computers, and they generate more "random" numbers than on one server.

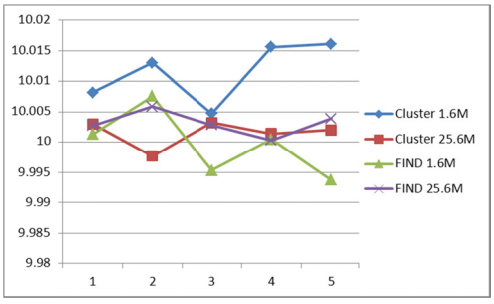

Figure 3: Variance comparison of each result.

We also compare the MapReduce program with a multi-threaded program. We run the multi-threaded program in 1.6M paths and 25.6M simulation paths, with separating to 1 (1.6M only), 2, 8, 16 and 32 threads. The multi-thread program will be run twice in each combination of parameters, except that 25.6M paths with 2 threads is run once. We record the running time, price, and variance, and choose the fast one as final result. It is shown in Table 1. From these results, when number of threads is more than the total processor cores (including Hyper-Threading) of the server, it cannot observably improve the performance. The server have two quad-core CPUs, equals 8 cores, so the running time from 8 threads to 32 threads is decreased little.

4.2 Stock Index Calculation on FIND

In this experiment, we test out the feasibility of FIND in data integrating, service workflow and continuous computing. The experiment also tests the usability and elasticity of FIND.

The dataset is the stock trade data retrieved from the web (http://www.sina.com.cn). A program is gathering updated trade data periodically, cleaning, extracting, transforming, integrating the data, and saving to HBase on FIND. There are 475,407 rows (counted at the end of this experiment), and about 32 attributes for each row. The new rows are appending continuously when the experiment is running. We choose the weighted sum index service, and give the list of 50 stocks and corresponding shares. The portal submits the corresponding service program and the list of stocks to the NameNode, then run it. The program iteratively calculates the newest index in background and saves it to HBase. Another user can visit a certain web page from the portal to view the index. We can regard the program as providing service and regard the web page as accessing service.

Figure 4 is a screenshot of the index query form. The square points are the index values calculated by the service. Because the auction trading runs from 9:30 to11:30 and from 13:00 to 15:00 in every trading day, the stock trade data is not updated after 15:00, and the value of the index is not changed too.

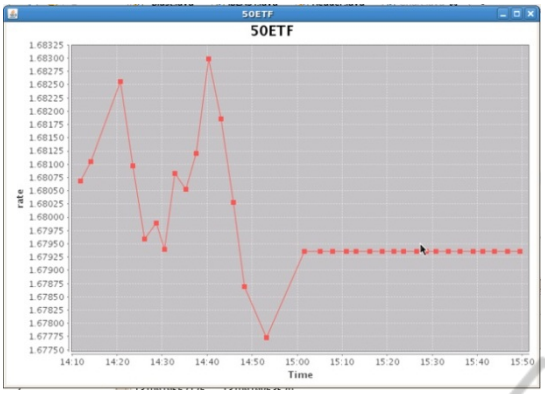

Figure 4: Screenshot of index query.

5 CONCLUSIONS

More algorithms and methods for dealing with large scale of data are presented in these years. The big data brings a data "Big Bang". It becomes important that how to discover patterns from the big data easily and quickly. In this paper, we present a data cloud platform for financial data services (FIND). It can collect and integrate financial data from various data sources, search and process information, analyze and mining patterns. FIND focuses on providing high performance computing capability, high quality integrated financial data and elastic data services. By using cloud computing technique, the FIND has capability and advantage on processing big financial data and can provide data services on demand. FIND helps users to use various sources of big data in an easy and a flexible way. The security and privacy of FIND need to study in the future work.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation Project of China under Grant (No. 61170096), Shanghai Leading Academic Discipline Project under Grant (No. B114), and National Key Technology R&D Program (No. 2012BAH13F02).

REFERENCES

Boyle, P., (1977). Options: A Monte Carlo approach. In *Journal of Financial Economics*, 1977, 4(3):323-338.

Buneman, P., (1997). Semistructured data. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 117–121. doi:10.1145/263661.263675

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber R. E., (2006). Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI'06)*, 205-218.

Chen, Y., Wang Y. Q. and Zhu Y. Y., (2006). Grid-Enabled Metropolis Shared Research Platform. In *Advanced Web and Network Technologies, and Applications*, Lecture Notes in *Computer Science*, 2006, Volume 3842/2006, 477-485. doi:10.1007/11610496_62

Cordeiro, R. L. F., Traina, C., Junior, Traina, A. J. M., López, J., Kang, U. and Faloutsos C., (2011). Clustering very large multi-dimensional datasets with MapReduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 690-698. doi:10.1145/2020408.2020516

Dean, J. and Ghemawat, S., (2004). MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design and Implementation (OSDI'04)*, 6(6):10.

Ghoting, A., Krishnamurthy, R., Pednault, E., Reinwald, B., Sindhwani, V., Tatikonda, S., Yuanyuan Tian and Vaithyanathan, S., (2011). SystemML: Declarative machine learning on MapReduce. In *2011 IEEE 27th International Conference on Data Engineering (ICDE)*, 231-242, doi:10.1109/ICDE.2011.5767930

Sul, S. J., Tovchigrechko, A., (2011). Parallelizing BLAST and SOM algorithms with MapReduce-MPI library. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, 481-489. doi:10.1109/IPDPS.2011.180

Vecchiola, C., Abedini, M., Kirley, M., Chu, X. C. and Buyya, R., (2010). Gene Expression Classification with a Novel Coevolutionary Based Learning Classifier System on Public Clouds. In 2010 Sixth IEEE International Conference on e-Science Workshops, 92-97. doi:10.1109/eScienceW.2010.24