Big Data - Financial Risk Computation Using Apache Hadoop (Map Reduce, HDFS)

Zeba Ahmad*, Apratim Rajendra**, Praveen Bhushan***

** IEEE Senior Member, Certificate in Risk Management - New York University,MCSD.Net, MCA, BCA*
*** IEEE Senior Member, Bachelor of Technology, Mechanical Engineering, Indian Institute of Technology Kanpur*
**** B.E REC Computer Science, Rourkela, India, Certificate in Advanced Calculus from City University of New York*

*zeba.ahmad7@gmail.com, **apratim.rajendra@gmail.com,*** praveen.bhushan@gmail.com,

Abstract- **With the growth of capital markets, the volume of financial data set that is being generated and consumed by various IT applications is growing at an exponential rate. The IT applications mentioned here could be of varying degree ranging from trading, risk analytic applications in front office; trade booking, clearing, and settlement applications from middle to back office. The increase in data set is not a major problem however the challenge is in its interpretation and analysis in a timely manner by various capital market firms, in order to execute profitable trading strategies and track risk PnL attributes. Hence there is a requirement of software solutions that could address issues of storing and analyzing the large scale volume of structured or unstructured data (of the order of Exabyte's a.k.a big data).Apache Hadoop is an open source software framework for storage and large scale processing of data sets. We have tried to explore the core components (MapReduce and HDFS) of Apache Hadoop and presented an approach to handle large scale data set that gets generated while computing risk generation of financial derivative securities.**

Index Terms- Big Data, Financial Risk, Apache Hadoop, Map Reduce, HDFS

I. INTRODUCTION

Big data is a term used to describe massive volume of structured or unstructured data that cannot be handled using database management tools or traditional data processing systems. Nowadays data is getting generated everywhere, from social networking posts, weather information, shopping and almost every activity. The challenge is to turn this massive data into information. There are challenges in being able to create, store, manipulate and manage this huge set of data. The big data could typically be order of petabytes (1,024 terabytes) or exabytes (1,024 petabytes). While the term may seem to reference the volume of data, it actually refers to the tools, processes and procedures to manage big data.

The amount of data in our world has been exploding and analyzing large data sets would become a key point for competition, innovation and productivity growth. E.g. Facebook users share billions of data content daily, this data can be used to understand users and their behavioral pattern and inclinations. Facebook reported in its third quarter of 2013 a 60 percent increase in revenues, with mobile now accounting for a major share in the company's advertising revenue. Facebook analytics chief Ken Rudin has confirmed the huge impact of big data on mobile advertising business..

II. STUDIES AND FINDINGS

A. Big Data Analytics:

Big data analytics refers to the process of collecting, organizing and analyzing large sets of data to discover patterns and extract useful information out of it. Generally these data are unstructured and loosely coupled. Big data analytics helps to correlate the data and to understand the information in the data. The primary goal of big data analytics is to help in making better decisions by analyzing the data. The technology can be used to decode data silos and revolutionize the growth and expansion of business in numerous ways E.g.

1. Social Networking sites can analyze the big data to predict user's personal preferences & could customize an ad feed to generate revenues based on it.
2. Movie streaming sites can go through the list of titles watched by viewers and can get a sense of the genres in demand.
3. Financial Industry can leverage the big data computing ability to price complex derivative instruments.
4. Courier companies can trace out the route of deliveries and could optimize it to save time and make it fuel efficient.

B. Comparison with Traditional Solutions:

- Relational Database Management System (RDBMS) – RDBMS stores structured data. Big Data on the other hand can handle both structured and unstructured data. The order of the data in RDBMS is typically of the order of the gigabytes. It cannot handle massive data due to the organization of data to preserve relationship. Relational data is often normalized & stored in that way. The cost of updating the data would be more in the RDBMS due to its structure. For a smaller set of data RDBMS would perform better than the Big Data. RDBMS is good for application where the dataset is continuously updated. In Big Data typically there is one write operation and multiple read operations.

- Grid Computing - The high performance grid computing also does parallel computation using a cluster of servers and a shared file system. It should be employed when the computation is not data intensive. Huge data transfer

could become a bottleneck due to the network delays. Big Data handles it by moving the computation to same server as where the data exists.

III. APACHE IMPLEMENTATION

Apache Hadoop is a framework for processing huge data sets on a cluster of computers. The framework is highly scalable and new nodes can be added as required to increase the computing power. It enables the distributed processing of the large data sets across cluster of servers. The two main components of the Apache Hadoop framework are:

1. MapReduce: The framework that manages workload and assigns job to the nodes in the cluster. It manages distributed data processing using the MapReduce programming paradigm.

2. Hadoop Distributed File System (HDFS): A distributed file system that provides high performance access to data stored across clusters.

1. MapReduce :

MapReduce is a programming model developed to support processing of large data sets(a.k.a. Big Data as discussed above).The concept though developed at Google is a combination of two simple functions as shown below-

1. Map: to map data set into a collection of <key, value> pairs
2. Reduce: to reduce overall pairs with the same key

For instance if we want to sum square of all the even numbers and odds numbers in a set of first n numbers MapReduce would look as shown below.

HDFS for storage and MapReduce for processing are the core components of Hadoop. They are coupled together in the sense that computation is done close to the server that has the relevant data. The map function as shown below is called in parallel and produces a pair such that key is the number and value its square. Even number keys and odd number keys are then grouped and are reduce to final value by summing up in the subsequent reduce functions.

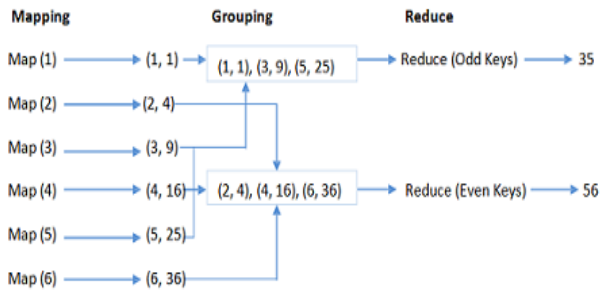

Figure 1

MapReduce Input and Output:

1. MapReduce operates exclusively on <key, value> pairs
2. Job Input: <key, value> pairs
3. Job Output: <key, value> pairs
4. Conceivably of different types
5. Key and value classes have to be serializable by the framework.
6. Default serialization requires keys and values to implement Writable.
7. Key classes must facilitate sorting by the framework.

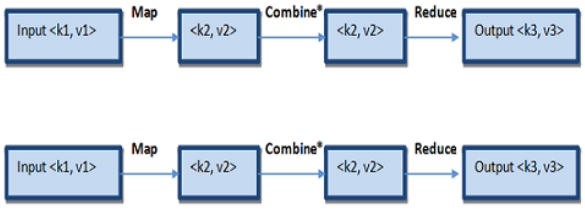

Figure 2

2. Apache Hadoop MapReduce Implementation :

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation is Apache Hadoop's MapReduce and in this paper we would like to explore the same. Apache Hadoop's cluster integrates MapReduce and HFDS and follows master/slave architecture such that Master Node/multiple Slave Nodes contains following nodes which are part of MapReduce layer.

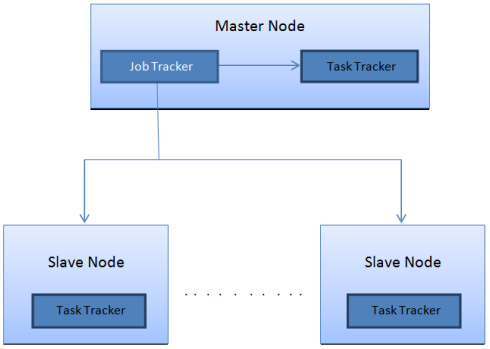

Figure 3 Map Reduce Layer

JobTracker - Single JobTracker per master

1. Responsible for scheduling the jobs' component tasks on the slaves
2. Monitors slave progress
3. Re-executing failed tasks Responsible for scheduling the jobs' component tasks on the slaves
4. Monitors slave progress
5. Re-executing failed tasks

TaskTracker - Single TaskTracker per slave execute the tasks as directed by the master. It executes the Mapper/ Reducer task as a child process in a separate JVM.

Map step
1. Master node takes large problem input and slices it into smaller sub problems; distributes these to worker nodes.
2. Worker node may do this again; leads to a multi-level tree structure
3. Worker processes smaller problem and hands back to master

Reduce Step - Master node takes the answers to the sub problems and combines them in a predefined way to get the output/answer to original problem

JobConf - It represents a Map-Reduce job configuration and is used to configure Mapper, combiner (if any), Partitioner, Reducer, InputFormat and OutputFormat implementations. JobConf is the primary interface for a user to describe a map-reduce job to the Hadoop framework for execution. The framework tries to faithfully execute the job as described by JobConf.

JobClient - It is the primary interface by which user-job interacts with the JobTracker. JobClient provides facilities to submit jobs and track their progress, access component-tasks, reports/logs; get the Map-Reduce cluster's status information and so on.adily find Mfiles related to your research work on internet or in some cases these can require few modifications. Once these Mfiles are uploaded in software, you can get the simulated results of your paper and it easies the process of paper writing.

3. HDFS Architecture:

HDFS is composed of a cluster of servers. The data is distributed across the cluster in blocks. It is based on the master slave architecture. The key components of HDFS are:
1. Name Node – It is a master server that contains information about the data nodes. It also manages file system operations and authorizes client's access to the data.

2. Data Node – These store the data as blocks within files. It creates, update and delete blocks based on the instructions from the name node.

Generally a dedicated server is for Name Node and all other servers in the cluster will have one data node. Name nodes are responsible for storing the metadata of the data nodes. When MapReduce queries for a data, it maps it to the corresponding data node. In order to gain performance, the data processing happens at the same place as the actual data storage. This is known as data locality based on the principle that moving computation is cheaper than moving data. The data nodes are responsible for serving read and write requests from clients. One drawback of this approach is the Single point of failure for the name node. If the name node server goes down then the whole cluster goes down.

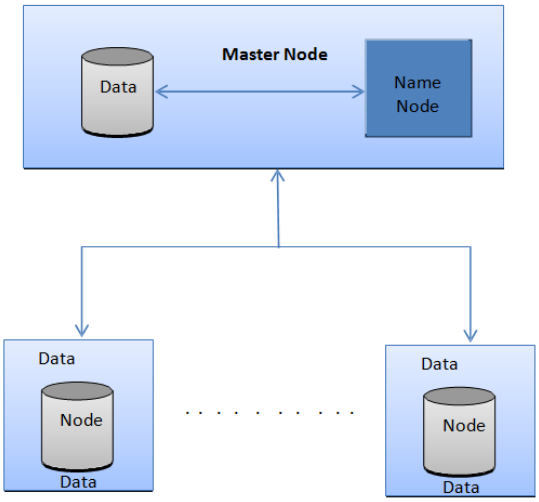

Figure 4 HDFS Architecture

A. Fault Tolerance & Data Storage Reliability - HDFS replicate data blocks for fault tolerance. The number of replicas can be configured during the file creation. In order to maintain the state of cluster, each data node sends a heartbeat message to the name server. If the heartbeat message is not received by a data node, it's marked as dead and no further requests are routed to it. The Name Node can initiate re-replication of the lost data.

B. Data Organization - HDFS is designed to support very large data sets and files. The typical block size in HDFS is of 64 MB i.e. the file is divided into blocks of 64 MB. A file creation request by client is first created into a temporary local cache. Once it reaches close to the block size then the client notifies the Name node, which subsequently creates a data node. When the file is closed the un-committed data in temporary cache is flushed to the Name node for persistence. The data replication is done through pipelining. For example if the replication count is two, then the client will receive the list of data nodes (for replication) from the Name node. When the first block is flushed to the data node, after committing the contents it transfers it to the next data node for replication. Finally the second data node will write the contents to its data repository.

C. Data Integrity and Security- HDFS ensures data integrity using checksum validations. The checksum of the files are stored in hidden files within the same namespace. The client can verify the contents by computing the checksum and validating it against the stored checksum. The authorization of the file permissions follows the same model as the Portable Operating System Interface (POSIX). There is an owner associated with every file and directory. It supports the read (r) and writes (w) permissions. The execute (x) permission means the user can traverse the directory structure.

IV. RESEARCH AND IMPLEMENTATION

A. Calculating Risk Parameters for Derivative Instruments Using Hadoop

Financial derivative securities are complex in nature and so is the market data associated with them. Modeling risk of such securities involves calculating various risk parameters like present value, Greeks (delta, Vega, gamma, theta etc.) in different scenarios. For instance how would the price of a derivative security fluctuate if its underlying does fluctuate? Now underlying can be simple stock or interest rate or volatility or any other derivative security or all of them. In this case we are talking about analyzing voluminous amount of market data that would include spot price, yield curve, volatility term structure etc. associated to respective underlying. A typical scenario could be that spots shift by 10% or yield curve bumps by 10 basis points or volatility fluctuates by 10% than how would the price of a given derivative security move. The set of scenarios can be even more complicated such that it includes multidimensional scenarios for example under a stock shift of 10% volatility also shifts by 10%.Each such shift requires in memory loading of spot prices and volatility structures, then doing the respective bumps and finally computing various risk parameters.

Imagine a portfolio of such derivatives securities containing tens of thousands of such complicated derivative securities and on which we want to compute various risk parameters under large number of scenarios. The modeling might become even more complicated and would require for extremely optimal ways of handling large data sets when it's done at the firm level. A firm can be a investment bank or asset management firm or any other large institutional investor. Such a firm, under normal scenario, would like to evaluate the profit and loss report for the securities that it has traded across all its trading desks when interest rate rises by 10 basis points. This would require the handling of humongous market data.

Apache Hadoops' Big Data framework hence forms the basis of addressing the issue in hand. Hadoops MapReduce is a software framework for easily writing applications which process vast amounts of data (of the order of multi-terabyte) in parallel on large clusters of commodity hardware in a reliable, fault- tolerant manner. A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the output of the maps, which are then input to the reduce tasks. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. The computational node and the storage nodes are the same i.e. MapReduce framework and Hadoops Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present; resulting in achieving optimal usage of bandwidth on the network or else data needed would transfer back and forth on the network.

As discussed in the section of apache Hadoop map reduce framework's core components the problem of calculating risk parameters of a portfolio of arbitrarily large number of securities can be effectively addressed using map reduce interfaces.

Let's take 10 derivative securities for example and try calculating delta under 2 scenarios where spotVolShift (spot shifts by 10%, volatility shifts by 10 %), only spot shifts by 15% and finally we need to report delta under each scenario for each of 10 instruments. Spot represents stock underlying and greek delta represents option price sensitivity with respect to underlying price.

1. It's a portfolio of 10 instruments each having its respective market data dependencies. Such that the market data is stored in the HDFS. Based on the sameness of market data dependency need to be shocked under a given scenario, instruments are clubbed together. So we form key as set of same market data dependencies shared by various instruments. Recalling our diagram from map reduce section Input's k1 = set of same market data dependencies for given set of instruments that needs to be shocked and v1 = set of scenarios defining the shock values..functions.

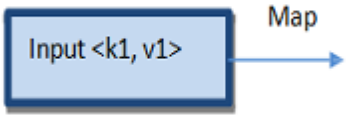

Figure 5

2. Using Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> (apache Hadoop's interface) implementation as shown below. This map phase does the parallel bumping of market data dependencies. The distribution of maps on individual nodes is done such that a given map has all the required market data on its respective node. This can be done using the notion of data locality as identified by apache HDFS name node server.

```
public class BumperMap extends MapReduceBase
implements Mapper<
MarketDataDependencyAndInstruments,List<Scenarios>,
MarketDataDependencyAndIns
truments,List<GranularScenario>>{
public void map
(MarketDataDependencyAndInstruments,List<Scenarios>
,..){
//This function bumps the market data as contained in
//MarketDataDependencyAndInstruments for each
scenario
//contained in the List<Scenarios>.So for example there
//are 2 scenarios spotVolShift and spotShift.
//Hence for spotVolShift both spot and vols are shifted
//by specified amount and bumped up
marketDataDependency is
//recorded and for spot shift only spot is shifted.
```

```
//MarketDataDependencyAndInstruments is updated with shocked
//marketDataDepencies.However the scenarios as obtained in
//List<Scenarios>, are translated into more granular list
//of scenarios List<GranularScenario> as required by the
//analytics library.
}
}
```

Where MarketDataDependencyAndInstruments represent a data holder for unshocked marketDataDependencies, shocked marketDataDependencies and list of instruments sharing such dependencies. Moreover the holder only contains the lightweight references to the actual market data and the data gets loaded whenever required. This way heavy network traffic can be avoided. Moreover GranularScenario represents analytics representation of more generic scenario. So a generic scenario is calculating delta when spot shifts by 10% however this could be translated to calculate difference of option prices between current spot and spot bumped up 10% which is equivalent to saying-

(OptionPrice2–optionPrice1)/underlying Price*0.1
Or in matrix form as
[optionPrice2, - optionPrice1] * [1/ underlying Price*0.1]

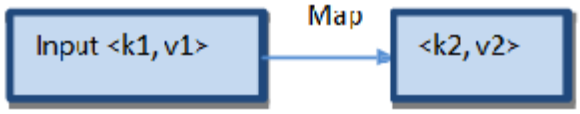

Figure 6

3. Now based on newly generated set of more granular scenarios a second map needs to be happening to execute similar scenarios in parallel. Hence as shown in item 1 the diagram gets extended as shown below
<k1,v1>= MarketDataDependencyAndInstruments,List<Scenarios>
<k2,v2>= MarketDataDependencyAndInstruments,List<GranularScenario>
<k3,v3>=MarketDataDependencyAndInstrument,Map<GranularScenario,Result>
Based on Granular Scenario analytics computes the risk parameters and in this case it computes option prices in the second map.

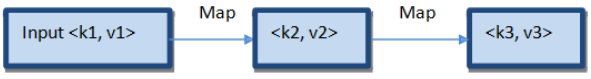

Figure 7

4. Now the reduction phase starts and using Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>(Apache Hadoop's interface)) we try collecting the results as shown below

```
public class ResultReduce extends MapReduceBase implements
Reducer<MarketDataDependencyAndInstruments,
Map<GranularScenario,Result>,Instrument,
Map<Scenario,Result> >{
public void reduce (MarketDataDependencyAndInstruments,
Map<GranularScenario,Result>,..){

//This function reduces the results by collecting scenario compute
//output of last map across various instruments.
```

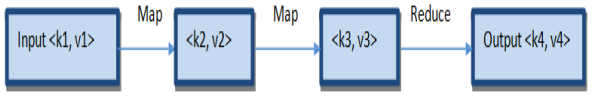

Figure 8

V. CHALLENGES

MapReduce has its own limitations. Apart from the most obvious issue where MapReduce is only suited to solving data problems where multiple tasks can be executed without any synchronization between the tasks there are other programming challenges that needs to be addressed.
As discussed in section 3 there are various cases where in varying degree of problems can come in the flow. Some of them are listed below-
1. One or more map tasks might fail and hence needs to be executed again. How to execute again such failed map tasks avoiding others which succeeded?
2. How to detect such failed tasks?
3. How to flag an incompleteness of an output generated out of reduce when one or more previous map tasks have failed?
4. How to reassemble the results of execution of failed map tasks with the final output generated in step 3?

Apache Hadoop's MapReduce framework provides solution to some of the problems as mentioned above via its JobTracker and TaskTracker interfaces however in order to provide a consistent output while detecting and flagging errors proactively requires additional software programming and it depends on sophistication level required by the clients consuming the output.

REFERENCES

I. David Turner, Michael Schroeck and Rebecca Shockley , Analytics: The real-world use of big data in financial services, May 2013
II. Michael Versace,Karen Massey, The Case for Big Data in the Financial Services Industry, September 2012
III. Mark Flood, H V Jagadish, Albert Kyle, Frank Olken, and Louiqa Raschid. Proc. Fifth Biennial Conf., Using Data for Systemic Financial Risk Management, Innovative Data Systems Research, Jan. 2011.
IV. An Oracle White Paper in Enterprise Architecture, August 2012

V. Amir Halfon, Big Data and Risk Management: Cracking the Code, July 02, 2012

VI. Steve Lohr, The Age of Big Data, New York Times, Feb 11, 2012

VII. Apache Hadoop, Hadoop.apache.org

VIII. http://en.wikipedia.org/wiki/Apache_Hadoop

IX. http://www-01.ibm.com/software/data/infosphere/hadoop

X. http://www.ibm.com/developerworks/library/wa-introhdfs/

XI. http://en.wikipedia.org/wiki/MapReduce

XII. http://www01.ibm.com/software/data/infosphere/hadoop/mapreduce/