DOI: 10.6245/JLIS.2016.422/653

Distributed Computing System and Big Data Real-time Processing Structure --Based on YARN, Storm and Spark

Wen-Ching Liou
Associate Professor,
Department of Management Information Systems,
National Chengchi University, Taiwan (R.O.C.)
E-mail: w_liou@nccu.edu.tw

Po-Wei Tseng
Software Engineer,
Taiwan Semiconductor Manufacturing Corp, Taiwan (R.O.C.)
E-mail: b5336789@gmail.com

Keywords: Apache YARN; Apache Storm; Apache Spark; Big Data Processing; Real-time Forecasting

---

【Abstract】

With the coming of the era of big data, the immediacy and the amount of data computation are facing with many challenges. For example, for Futures market forecasting, we need to accurately forecast the market state with the model built from large data (hundreds of GB to tens of TB) within tens of milliseconds.

In this paper, we will introduce a real-time big data computing architecture to resolve requests of high speed processing, the immense volume of data and the request of large data processing. In the meantime, several algorithms, such as SVM (Support Vector Machine, SVM) and LR (Logistic Regression, LR), are implemented as a subproject under the parallel distributed computing system. This architecture involves three main cloud computing techniques:

1. Use Apache YARN as a system of integrated resource management in order to apply cluster resources more efficiently.
2. To satisfy the requests of high speed processing, we apply Apache Storm in order to process large real-time data stream and compute thousands of numerical value within tens of milliseconds for following model building.

3. With Apache Spark, we establish a distributed computing architecture for model building. By using Spark RDD (Resilient Distributed Datasets, RDDs), this architecture can shorten the execution time to within hundreds of milliseconds for SVM and LR model building.

To resolve the requirements of the distributed system, we design an n-tier distributed architecture to integrate the foregoing several techniques. In this architecture, we use the Apache Kafka as the messaging middleware to support asynchronous message-based communication.

Introduction

The Background of the High-Frequency Trading System

The requests of big data nowadays are stronger with the proportion of high frequency trading (HFT) springing up. According to the data released by the New York Times in 2012, there are 50% HFT in all US securities trading market. Also, the ratio of HFT is up to 60-73% in 2009. Because of short-term trends of market and ultra-fast quotes, people are hard to decide when to buy or sell in time; moreover, there are too many external conditions to evaluate. As a result, we design an efficient n-tier distributed computing architecture to help us to resolve these difficulties.

When the securities dealers implement an HFT system, the calculation speed is one of the most critical requirements. AN HFT system needs to process market quotes very fast because the variation of HFT market price trends is fleet. It must calculate market state quickly and predict market trends in time. Then, place an order with lightning speed and earn slight benefits. Many winning orders earn a profit.

On the other hand, big data processing is also an important requirement. Before an HFT system makes a forecast for the market price, it needs to collect the market quotes from the market and then process the immense price data.

The HFT systems make complicated Technical Analysis (TA) of large price data. Then, the strategy-planning subsystem receives the Technical Analysis result and does considerable calculation with complex algorithms. The third requirement of implementation of an HFT system is big data storage. For the enormous amount of data (millions of market state per second for all futures), an HFT system usually needs to continually store to the database and fast fetch data in the following machine learning algorithm model building.

Computers replace humans in the HFT systems, so the decisions on market trends are more rapid and accurate in most cases. Thus the retail investors usually are suppressed by the securities

dealers who have plenty of resources. The unfairness of information, equipment and other resources cause plunders of the retail investors. That is one of the reasons why we investigate the HFT architecture. Eventually, we want to resolve foregoing three main requirements for establishing an HFT system by the integration of distributed computing techniques and big data processing techniques.

Design the n-tier Distributed Computing Architecture

For satisfying the foregoing requirements in HFT system, this paper designs an n-tier distributed architecture by these cloud computing techniques: Apache Storm, Apache Spark, Apache YARN and Apache Kafka.transmission. We will introduce the n-tier architecture in detail on the following sections.

In concept, the State Center is all the foundations of the HFT systems. It needs to calculate market states in several milliseconds to tens of milliseconds. This subsystem can receive continued market quotes and pass them to multiple nodes in cluster by the implementation of Apache Storm. The Plan Center is accomplished by Spark RDD (Resilient Distributed Dataset) and Spark Machine Learning library (MLlib), it fetches needed TA results from Hadoop Distributed File System (HDFS) and then build strategies of investment in extremely hundreds of milliseconds for detecting short-term trends. The Trade Center is responsible for detections of real-time market trends by execution of the machine learning models (or strategies) built from the Plan Center. After running the models, the Trade Center receives the calculated market states from the State Center and immediately sends the orders to the Order Center if signals occur from models.

Furthermore, we use Apache YARN to make the unified management of all services for the most efficient use of cluster resources. And, Cloudera offers a web UI and is used to manage for the Hadoop related services. We are going to describe these parts in the following sections.

Related Works

The N-tier Distributed Computing Architecture

In this paper, we focus on a specific n-tier distributed computing architecture for the HFT systems, which is scalable, fast, loosely coupled and asynchronous. In addition, different from traditional n-tier distributed architectures, the HFT systems need to process large data sets in a short time. On the other hand, these communications between nodes are very complicated. As a result, we need a unified messaging middleware for the transmission of messages between every service.

Many computing architectures are designed for specific framework or usage. In (Manuel & AlGhamdi, 2003), data-centric design for n-tier architecture is mentioned, which is suitable for .Net or J2EE framework specially. Also, they encourage a software process that reduces the development time considerably. In (Ding, Hu & Sun, 2008), inspired by immune system, they attempt to improve the existing distributed object computing (DOC) infrastructure. For better efficiency, the HFT architecture has to be specially optimized for complicated and high-frequency communications.

Besides, "Loosely Coupled" is also a critical requirement of this architecture. In (Joshi, 2007), the authors introduce a loosely-coupled real-time distributed system and the key challenges in building next-generation distributed systems. Based on this paper, a distributed system needs to involve a messaging-oriented middleware to achieve "Loosely Coupled".

To sum up foregoing researches, for a high-speed distributed system, a messaging system is very critical. Thus, we use Kafka as the messaging middleware of our HFT system for its good performance and loosely coupled interfaces. Nevertheless, inside the real-time business logic tier in our HFT system, we use Netty instead.

High-Throughput Distributed Messaging System——Kafka and Netty

Because of the trading is high-frequency, we need a high-speed messaging middleware. There are some researches about the messaging systems in distributed computing architectures. In (Kreps, Narkhede & Rao, 2011), the authors develop Kafka as the log processing system. According to that research, Kafka has much higher throughput than conventional messaging systems. Also, LinkedIn uses Apache Kafka to transfer their events in the Hadoop ecosystem (Sumbaly, Kreps & Shah, 2013). For the much higher throughput and the fast processing speed, we used Kafka in our architecture. We not only use Kafka as our log processing system but also use it as the critical messaging middleware in our HFT architecture.

In our system, we divide the services into two tiers, real-time business logic tier and non-real-time business logic tier. In the real-time business logic tier, we use an ultra-fast messaging system in the Storm topologies, named Netty. Netty is built in Storm and transmits messages faster than Kafka inside the Storm topologies in our experiments. Originally, Netty is an independent framework which is contributed by Yahoo engineers. Based on the official website, Netty is asynchronous, high-performance, low resource consumption and low latency Netty (http://netty.io/index.html).

On the other hand, TIBCO FTL is also a high-performance messaging middleware. According to their official website, they introduce the FTL messaging middleware platform for real-time,

high-throughput data transmission TIBCO (http://www.tibco.com/).

. FTL is also suitable for messages transmission in the HFT system.

Real-time Streaming Data Processing——Storm

Apache Storm is a free and open sourced distributed real-time computing framework. By Storm framework, we can build a processing system to receive continuous data stream (is real time quotes in HFT system) and calculate numeric values per second in time. The framework is scalable, resilient, extensible, efficient and easy to use by administrators (Toshniwal et al., 2014). Storm supports many use cases, such as real-time analytics, online machine learning, continuous computation, distributed RPC (Remote Procedure Call), ETL (Extract-Transform-Load), and more. Storm community becomes maturing and many companies use it and the most famous users contain social media website Twitter, the raised sharply company Alibaba in China and search engine champion Baidu in China. Twitter use Storm as their twit processing system initially and then they open-sourced the project. Storm provides great efficiency for real-time computation in large data stream. It can handle over one million tuples per second per node Apache Storm (https://storm.apache.org/). More importantly, Storm integrates with some existing technologies, such as conventional messages queue systems, distributed computing techniques and database systems like Kafka, Apache Hadoop and HBase (Jones, 2013).

In fact, we apply Storm to process continued tuples stream to complement the original Hadoop architecture in the HFT system. Previously, we use Hadoop Map/Rreduce framework to process big data set with batch jobs in many situations. When the requests of processing streaming data occur, Storm becomes more popular than Hadoop Map/Rreduce framework on some specific cases. In the HFT systems, because the Map/Rreduce programming model is difficult to handle the streaming market quotes, Storm stream data processing system is involved in the HFT architecture.

A Storm job, or more specifically a Storm topology, contains two classes of components: spout and bolt. The function of spout is receiving tuples from external sources (Ex. specific Kafka topics) or another topology and passing the tuples to bolts which need it. Figure 1 shows a simple Storm topology that processes Tweeter articles and counts the words in the data stream. The Storm data processing system involves streams of data sets (or said tuples) flowing through topologies (Toshniwal et al., 2014). Initially, the streams flow from queues, such as Kafka, to bolts according to programmers setting. Programmers can write any logic on every bolt. When the data stream flows through the bolts on the topology, the bolt which receives tuples runs specific logic with every tuple.

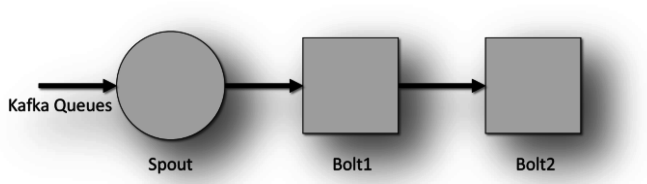

Figure 1 **A Simple Topology**

Figure 2 shows high level architecture of Storm. There are three roles that are related to Storm: Nimbus, Supervisor and Zookeeper (not belong to Storm) (Toshniwal et al., 2014). Generally, the Storm is a master-slave architecture that has one master (called Nimbus in Storm) and many slaves (called Supervisor in Storm). The Nimbus takes the topology jar from client program and is responsible for overall status management of the Storm topology.

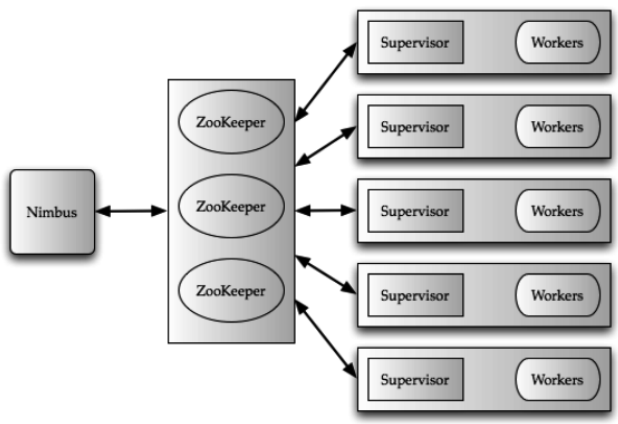

Figure 2 **High Level Architecture of Storm (Toshniwal et al., 2014)**

In our system, we use Storm to process real-time market quotes and calculate market states in time. By Storm, we can compute millions of market states in several milliseconds. We will describe this part in detail in the following sections.

Forecasting Model Building——Spark

When training a predictive model of investment, we use Apache Spark RDD to speed up the processes. According to (Zaharia, 2010), Spark is a very useful framework to process large data sets. That research introduces a concept called Resilient Distributed Datasets (RDDs). An RDD is a read-only dataset across a set of machines that is fault-tolerant. By Spark RDD, the speeds of the HFT systems are 10 times faster than the version of Hadoop Map/Reduce in iterative machine learning jobs (Zaharia, 2010).

The HFT systems use Spark on machine learning model building over HDFS data. In (Zaharia, 2012), the authors introduce Spark as an analysis tool over the Hadoop data. Based on their research, Spark can access any Hadoop-supported storage system that makes it to complement Hadoop for large-scale data analysis. That is very critical for the HFT systems because we can access large data sets of market states from Hadoop-based storage and use it in memory. Thus, the run time of machine learning jobs will be reduced outstandingly and the consumption of memory is lower.

Besides running machine learning jobs by Spark, this framework can also be used to process streaming data. In (Zaharia, 2012), by Spark streaming, the authors propose an online Expectation-maximization algorithm, which updates the state of car traffic in the San Francisco Bay Area from lots of GPS sources within a small number of seconds.

Resource Management——Cloudera and YARN

With the increased usage of distributed computing techniques, the resource management of the HFT systems becomes important. According to (Buyya, Broberg & Goscinski, 2010), they introduce Cloudera as one of the major enterprise solutions based on Hadoop. By Cloudera Hadoop Distribution, Hadoop can be deployed and installed easily on clusters. Otherwise, Cloudera offers an administration tools that users can use it to configure everything they need (Buyya, Broberg & Goscinski, 2010). Because the HFT systems use HDFS, HBase and Zookeeper, we choose Cloudera to manage these distributed services more easily.

On the other hand, as mentioned above, we use some Hadoop services in the HFT systems. And, in (Vavilapalli, 2013), two critical shortcomings are exposed:

1) The Map/Reduce programming model is tight coupling and forces developer to abuse it.
2) The jobs' control flow is centralized, which caused scalability concerns for the job scheduler.

According to (Vavilapalli, 2013), YARN provides: greater scalability, higher efficiency, and enable lots of different frameworks to efficiently use the cluster resources (Vavilapalli, 2013). Thus, we can use YARN to integrate a large number of services that includes Storm cluster, Kafka brokers and other supporting services.

System Architecture

Overview of the Distributed Computing HFT System

The HFT system runs on a distributed cluster and contains several main subsystems and services. These subsystems are responsible for the most critical functions, such as computation of market states, machine learning algorithm model building or running models to forecast future trends. Furthermore, other supporting functions are also important but not the main subjects of the HFT system.

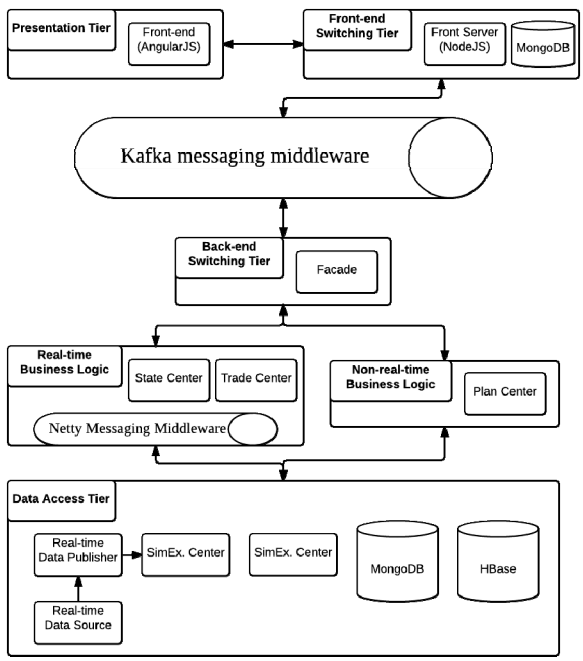

Figure 3 **The N-tier Distributed Architecture of the HFT System**

We arrange the whole system into an n-tier distributed architecture: Presentation Tier (PT), Front-end Switching Tier (FST), Back-end Switching Tier (BST), Real-time Business Logic Tier (RBLT), Non-real-time Business Logic Tier (NRBLT) and Data Access Tier (DAT) as shown in Figure 3. This architecture is evolved from the three-tier distributed architecture (Aarsten, Brugali, & Menga, 1996) (Hirschfeld, 1996). Eventually, we separate the Business Logic Tier into Real-time Business Logic Tier and Non-real-time Business Logic Tier. Besides, we use two levels messaging middleware to resolve the high-frequency requirements in the whole system. The functions of each tier are described as follows:

I. Presentation Tier (PT)

This tier fetches the data from BLT and prepares web pages to present for users browsing online. For speeding up the loading rate and lowering the latency of access time, the Node.js (https://nodejs.org) web server can access the cached MongoDB (said Front Cached Database) when it needs small-size data or usual user data, such as user information and website settings.

For unifying the communications with back-end cluster, we use a Façade service to handle all the requests from users to back-end cluster. All front-end services in PT cannot connect with back-end services or receive messages from Kafka queues besides the Façade service. This design makes our architecture loosely coupled and gives it maintainability and scalability.

In the users' webpages, we develop the web components by AngularJS framework. And, we deploy the front server on Node.js (https://nodejs.org), which is high-performance, scalable and very suitable for transmission of high-frequency requests.

II. Front-end Switching Tier (FST)

The Front-end Switching Tier receives web requests and passes them to Façade through Kafka messaging system. This tier contains a front server that is deployed on Node.js (https://nodejs.org). For efficiency reason, if the request data are stored in MongoDB, the front server accesses the data directly from MongoDB by itself.

All front requests must be dispatched to Kafka through front-end switching tier, and not allow being thrown to the back-end cluster.

III. Back-end Switching Tier (BST)

The Back-end Switching Tier fetches the messages from Kafka, and transmits the strategies signals and market states to front-end. Note that the BST is the unified entry interface from front-end to back-end. All requests from front-end server need to be transferred through the back-end switching tier.

IV. Real-time Business Logic Tier (RBLT)

This tier is a critical part of the whole HFT system, which is mainly responsible for real-time data processing and calculation. It contains two important services: State Center and Trade Center. State Center is a Storm topology that processes streams of real-time quotes promptly to calculate millions of market states per second. The completed market states are passed to Trade Center as well as stored into HBase persistently. HBase is built on the HDFS, which is used for storing

market states that are calculated by State Center. In the meantime if Trade Center has running machine learning models for forecast of market trends, the running models calculate signals of buy or sell possibly. Figure 4 describes this tier further.

Originally, the State Center and the Trade Center are separated into two topologies. The State Center calculates market states and passes them to the Trade Center through Kafka messaging system. However, for better efficiency and higher speeds, we merge these two topologies and replace the Kafka messaging middleware with Netty, which is around 10 times faster than Kafka in our Storm topology. The Netty is built in Storm, which helps us with the high-frequency transmission of messages.

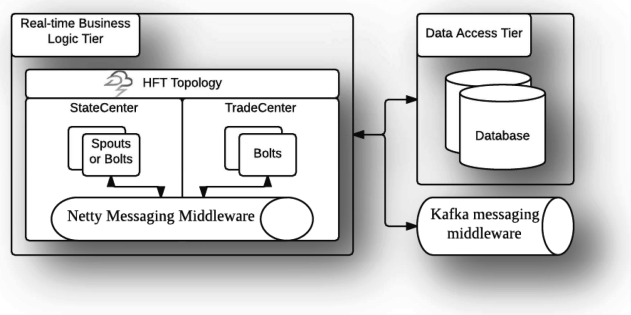

Figure 4 **Real-time Business Logic Tier in Detail**

V. Non-real-time Business Logic Tier (NRBLT)

Users can make strategies for specific futures products by using market state data on the Plan Center anytime. The strategies are stored into MongoDB for fast accessing from the front-end Node.js server. R Academy offers a platform for users to design their R programs and test it on the Trade Center online. This service combines the R programs and Spark RDD to provide an interface to access large data sets fast.

VI. Data Access Tier (DAT)

The DAT is used to access all data from the database or outer data sources. All processed data are sent to the DAT that stores the big data persistently with low latency. Also, the DAT offers an order interface to connect with the external securities dealers for transmission the user orders. On the other hand, the DAT combines several market quotes every ticks into one K-Bar every seconds.

In addition to above four tiers, the HFT system includes the unified messaging middleware that is implemented by Kafka. The messaging system of the HFT system is critical because the transmission of messages is high-frequency.

This architecture is pretty complicated and delicate. We implement many cloud computing technologies and we will describe them in following sections.

Lightning Calculation for Market States and Low Latency Storage——State Center

State Center is in charge of calculation of real-time quotes and saving the result market states into database. In order to retain a span (at least 500 milliseconds) for following machine learning model execution, it must satisfy the requests of fast calculation that is limited to tens of milliseconds (exclude transmission overhead time).

Originally, we implement State Center by Storm as a single topology in order to calculate market states fast. Figure 5 shows the whole State Center topology initially. In the topology, the KafkaSpout receives real-time quotes from the external RealtimeDataPublisher service that subscribes for real-time quotes from true market and continually sends the tick prices through the distributed messaging system, Kafka. The KafkaSpout then passes the prices to following 18 ComputeStateBolt. Each ComputeStateBolt carries different ComputeStateLogic and uses it to calculate the market states that are defined in the specific TA logic. The 18 ComputeStateBolt then send the result market states to specific TA WriteDataBolt. Note that every WriteDataBolt writes corresponding TA data into HBase. For example, The MA (Moving Average, is a technical analysis tool) ComputeStateBolt sends market states to MAWriteDataBolt that exclusively stores MA market states. Outside the topology, all black lines are transmitted by Kafka distributed messaging system. Besides, Netty is the messaging channels inside the Storm topology.

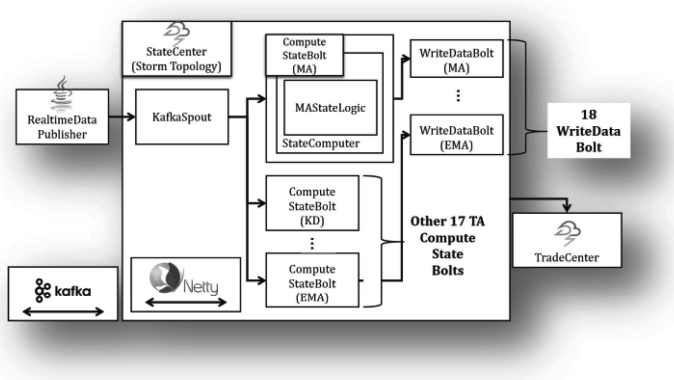

Figure 5 **State Center Topology**

Build Model Fast Over Big Data Sets——Plan Center

The investment strategies, or said investment models, are very critical for HFT system. Plan Center takes charge of the creation of investment strategies. By machine learning algorithm, Plan Center uses historical market data to build investment strategies. Also, it provides many customized algorithm parameters for users. There are two requirements of the created strategies we need to satisfy in HFT system:

1. **Fast**: We need to create the models before the trend changes because the trends are so transient and easy to miss in high-frequency trading.
2. **Big data loading in short time**: When Plan Center runs machine learning algorithm to create investment strategies, Plan Center needs to load large historical data sets from HBase and analyze them simultaneously in a short time.

To solve the fast and large data sets problem, we implement Plan Center by Apache Spark framework. By Spark RDD, Plan Center can load hundreds of gigabytes market states data into memory and analyze them across many nodes in cluster (Zaharia, 2012) (Zaharia, 2010).

Plan Center offers several machine learning algorithms for the creation of investment strategies, such as Support Vector Machine (SVM), Logistic Regression (LR) and Classification.

Forecast Market Trends Rapidly and Accurately——Trade Center

After the investment strategies are produced, users can choose the investment strategies that they want to use on web pages. The original architecture of Trade Center is shown in Figure 6.

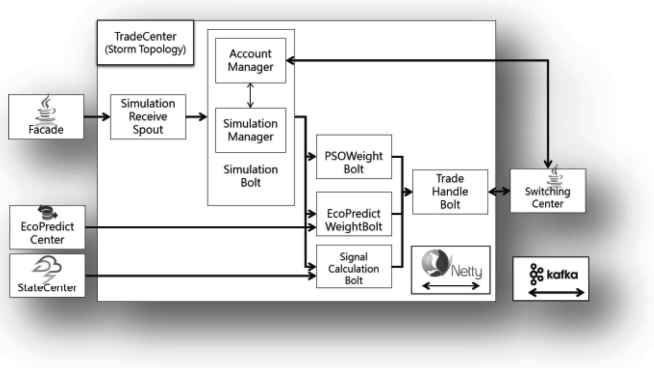

Figure 6 **The Original Design for Trade Center**

The Integration of State Center and Trade Center

The cost of time for pulling data from Kafka queues is extremely short and below tens of milliseconds commonly. But the Netty messaging system inside the Storm has much more rapid transmission speeds than Kafka in usually several milliseconds from one vertex to another. In order to reduce the overhead time of market states transmission, we combine State Center topology and Trade Center topology into one large topology, named HFT system.

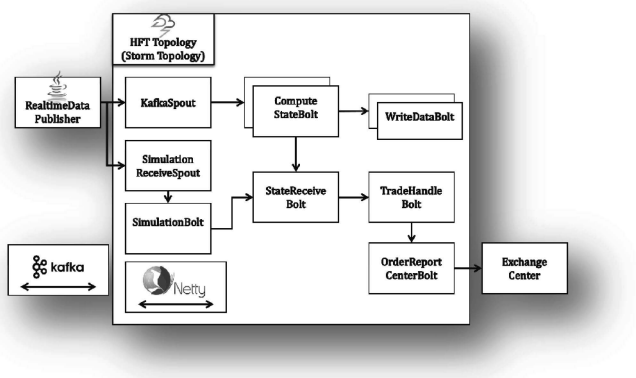

Figure 7 **The Architecture of the HFT Topology**

The HFT system actually is a large topology that pulls data from Kafka queues and writes data into HBase and MongoDB. Figure 8 shows the architecture of whole HFT topology. By the combination of State Center and Trade Center, the cost time of messages transmission is reduce to several milliseconds. The complete experiments data are described in detail on section of Experiments.

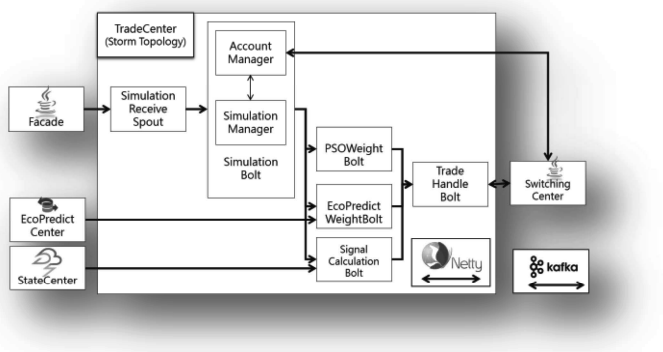

Figure 8 **The Architecture of the HFT Topology**

Cluster Resources Management

Because of the complexity of the HFT architecture, we use YARN to launch most services and manage all resources on cluster. Also, we monitor the health of each node and Hadoop service on Cloudera. We can customize the configuration of Hadoop services, monitor our services status on web pages and operate the cluster services easy (Hansen, 2012). The comparisons of optimized configurations for the HFT system are described in next section.

Experiments

In the previous section, we introduced the requirements of the high frequency trading and the detail of the n-tier architecture that used on the high-frequency trading. In this section, we present the experimental result to prove that the architecture's performance is good enough for high-frequency trading.

Experimental Environment

For evaluating the architecture's performance, this paper proposed several experiments. Because of the high-frequency and real-time requirement, this system must have the ability that handles considerable requests in a very short time. Specially, on State Center, we need to calculate millions of market states within one second. As a result, this paper specifically designs experiments for the State Center. We will compare the results between different numbers of futures and figure out how powerful this architecture can be. To implement the experiments, we prepare 8 computers as the cluster and 6 of them as the supervisors to run the Storm topology. More information of execute environment is shown in Table 1.

Table 1 **The detailed information of the cluster**

Host	CPU	RAM	HDD	OS	Storm Role	Zookeeper	Kafka
nccu-master	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz	7.7 GB	917.6 GB	Ubuntu 14.04	Storm UI	-	-
nccu-mgmt	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	3.8 GB	458.8 GB	Ubuntu 12.04	Storm Master	-	-
nccu-n01	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	3.8 GB	459.1 GB	Ubuntu 12.04	Storm Supervisor	-	-
nccu-n02	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	3.8 GB	459.1 GB	Ubuntu 12.04	Storm Supervisor	-	-
nccu-n03	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	3.8 GB	459.1 GB	Ubuntu 12.04	Storm Supervisor	-	-

(continued)

Table 1 *(continued)*

Host	CPU	RAM	HDD	OS	Storm Role	Zookeeper	Kafka
nccu-n04	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	3.8 GB	459.1 GB	Ubuntu 12.04	Storm Supervisor	Follower	Broker
nccu-n05	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	3.8 GB	459.1 GB	Ubuntu 12.04	Storm Supervisor	Leader	Broker
nccu-n06	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	3.8 GB	459.1 GB	Ubuntu 12.04	Storm Supervisor	Follower	Broker

To test the extreme efficiency of this architecture and find out the most appropriate configuration for the cluster, we compare the average computing time of all market states for each experiment. We add a bolt into the architecture to collect metric data of the market states computation.

Implementation of the Experiments

In order to implement this analysis, we add a new bolt, named ExpStateReceiverBolt, into our original topology to collect all computing metric data of market states. While all market states of one k-bar arrived, this bolt will sum the accumulated calculation time of these k-bars and calculate the average value. Figure 9 shows the performance result and Figure 10 shows the numbers of market states under N futures.

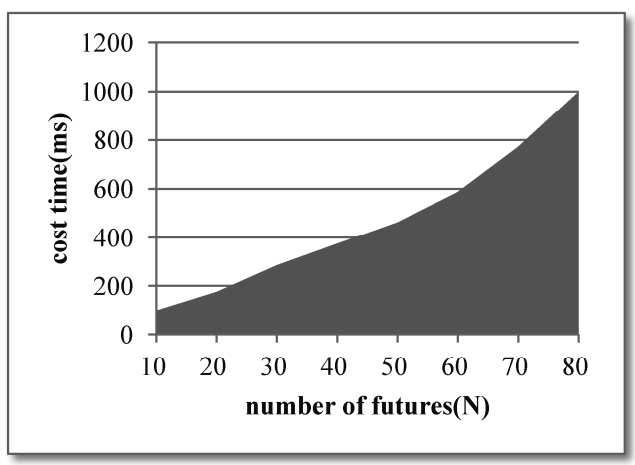

Figure 9 **The average calculation time of market states computation for N futures**

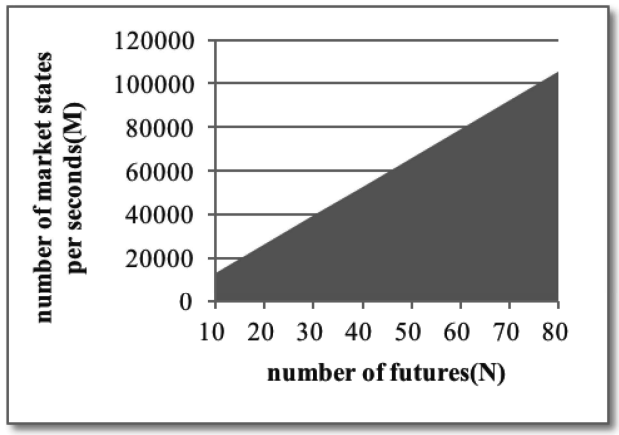

Figure 10 **The number of market states are computed for N futures per seconds**

If the time granularity is one-second, the bolt will receive 1,318 market states for each future per seconds. If the number of futures is 50, the State Center needs to calculate 65,900 market states for each second. The ExpStateReceiverBolt will compute the average cost time of the market states computation. The calculation time in the HFT system must be under 1,000 milliseconds and had better to be under 500 milliseconds because the HFT system still has to send the orders to exchange center after the computation of market states.

Low Latency for Computation

In Figure 8, we can see the average cost time for each k-bar's market states computation. When the number of futures is 10, which means the HFT system processes 10 futures' quotes at the same time, the average cost time of market states computation is 97.7 milliseconds. While the number of futures is 20, the average cost time is 175.0 milliseconds. There is approximately a linear relationship between the average cost time and the number of futures. If the number of futures is 80, the average cost time of market states computation is 998.89 milliseconds. As a result, with 6 supervisor (in Storm, supervisors are responsible for the logic execution), the HFT system can handle 80 futures' market states computation and the latency is under 1 second (1,000 milliseconds). Theoretically, if we want to support 1,000 futures and do not delay than 1 second, we need 75 supervisors to compute the market states. If we want to decrease the latency to 500 milliseconds, we should have 150 supervisors to support the computation.

One of the critical parts on low-latency computation of the State Center is the implementation of the Storm framework. The State Center is built on top of the Storm framework and benefit the

State Center on streaming quotes processing. Also, with Netty messaging system built in the Storm, the Compute State Bolts emit 105,440 market states per seconds with 998.89 milliseconds latency. The latency of our architecture is pretty low. In the situation of **6 supervisors and 50 futures**, the cost time of all computation of market states is **below 500 milliseconds**.

Conclusion

In this paper, we introduce a distributed, high frequency, low latency and loose coupling architecture based on several distributed computing technologies, such as Storm, Spark, Kafka and YARN. By this architecture, we implement a high-frequency trading (HFT) system, which contains several tiers that each tier is responsible for different function:

1. The presentation tier displays information, receives user's requests and passes requests to back-end cluster.
2. The front-end switching tier receives all the requests from front-end and passes them to back-end switching tier.
3. The back-end switching tier fetches front-end requests and dispatches them to right services in business logic tier.
4. The most important tier, the business logic tier, needs to calculate market states, build trading strategies and handle user requests.
5. If services need to access database or outer data sources, they will get the data through the data access tier.

By the high-performance messaging middleware, Apache Kafka, the services of the HFT system can send considerable messages to each other.

This architecture gives a general solution for high-frequency distributed computing system in real time. In the business logic tier, the state center and the trade center are very critical because they need to process high-frequency quotes and trading orders in a very short time (under hundreds of milliseconds). By implementing the Apache Storm framework, they have great performance that can handle millions of market states calculation and order submissions within tens of milliseconds to hundreds of milliseconds.

Otherwise, the n-tier architecture of the HFT system is highly scalable. For example, if we want to enhance the front-end server's performance, we just add some node.js server into front-end tier. If we want to support much more calculation of futures' market states, we just easily add some

machines into Storm cluster in the business logic tier. With the high scalability of the architecture, we can calculate the market states in tens of milliseconds as long as we have enough machines. Also, the performance of the HFT system is excellent based on experiments above. Under normal circumstances, this system can support the market states calculation of 80 futures with just 6 machines and do not delay than 1 second. The performance of this HFT system based on the n-tier architecture is good enough to forecast the short trend in high-frequency trading market.

The n-tier distributed architecture is suitable for many use cases, such as high-frequency data processing, considerable volume of data computation and large data set model building. We proposed a two-tiers messaging system to handle the message exchanges of the real-time and non-real-time data processing subsystems at the same time. This paper proved the performance of the real-time streaming data processing and the high-frequency quotes computation in the n-tier architecture. In the future, we will add more factors and keep on testing much more detailed configuration for cluster to improve the performance of the HFT system. And, to accomplish the following goals:

1. Build a distributed system that can support the market states computation of all futures in Taiwan and reduce the latency to several milliseconds to tens of milliseconds.
2. Shorten all messages' length in whole system by encoding the messages to decrease the bandwidth usage.

This paper try to combine several big data techniques to an n-tier distributed architecture for resolving high frequency data processing problems. We implement this architecture on a high frequency trading system and the high frequency system processes millions of market states within hundreds of milliseconds. Also, this architecture can be used on many other high frequency, big data, and streaming data processing system.

Acknowledgements

This paper is partially supported by TSMC Campus Collaborated Plan and partially supported by the "The Cloud for the Strategy Trading" project with Chunghwa Telecom.

References

Aarsten, A., Brugali, D., & Menga, G. (1996). *Patterns for three-tier client/server applications.* Paper presented at the Proceedings of Pattern Languages of Programs (PLoP), USA.

Apache Storm. Retrieved from https://storm.apache.org/

Buyya, R., Broberg, J., & Goscinski, A. M. (Eds.). (2010). *Cloud computing: Principles and paradigms* (Vol. 87). John Wiley & Sons.

Cloudera. Retrieved from http://www.cloudera.com/content/cloudera/en/home.html

Ding, Y. S., Hu, Z. H., & Sun, H. B. (2008). An antibody network inspired evolutionary framework for distributed object computing. *Information Sciences*,*178*(24), 4619-4631.

Hansen, C. A. (2012). *Optimizing hadoop for the cluster.* Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.486.1265&rep=rep1&type=pdf

Hirschfeld, R. (1996). *Three-tier distribution architecture.* Paper presented at the Proceedings of Pattern Languages of Programs (PLoP), USA.

Jones, M. T. (2013). *Process real-time big data with Twitter Storm*. IBM Technical Library.

Joshi, R. (2007). *Data-Oriented Architecture: a Loosely-Coupled Real-Time SOA*. CA: Real-Time Innovations, Inc.

Kreps, J., Narkhede, N., & Rao, J. (2011, June). *Kafka: a distributed messaging system for log processing.* Paper presented at the Proceedings of 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece.

Manuel, P. D., & AlGhamdi, J. (2003). A data-centric design for n-tier architecture. *Information Sciences*, *150*(3), 195-206.

Netty. Retrieved from http://netty.io/index.html

Node.js. Retrieved from https://nodejs.org

Sumbaly, R., Kreps, J., & Shah, S. (2013, June). *The big data ecosystem at LinkedIn*. Paper presented at the Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, New York, New York, USA.

TIBCO. Retrieved from http://www.tibco.com/

Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., . . . Ryaboy, D. (2014). *Storm@twitter*. Paper presented at the Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, Utah, USA.

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... & Baldeschwieler, E. (2013, October). *Apache Hadoop YARN: Yet another resource negotiator*. Paper presented at the Proceedings of the 4th annual Symposium on Cloud Computing, Santa Clara, California.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I. (2012, April). *Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing.* Paper presented at the Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, San Jose, CA.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., ... & Stoica, I. (2012). Fast and interactive analytics over Hadoop data with Spark. *USENIX Login*, *37*(4), 45-51.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010, June). *Spark: Cluster computing with working sets.* Paper presented at the Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, Boston, MA.

分散式計算系統及巨量資料處理架構設計－基於 YARN，Storm 及 Spark

劉文卿
國立政治大學資訊管理學系副教授
E-mail: w_liou@nccu.edu.tw

曾柏崴
台灣積體電路製造股份有限公司軟體工程師
E-mail: b5336789@gmail.com

關鍵字：Apache YARN；Apache Storm；Apache Spark；巨量資料處理；即時預測

【摘要】

近年來，隨著大數據時代的來臨，即時資料運算面臨許多挑戰。例如在期貨交易預測方面，為了精準的預測市場狀態，我們需要在海量資料中建立預測模型，且耗時在數十毫秒之內。

在本研究中，我們將介紹一套即時巨量資料運算架構，這套架構將解決在實務上需要解決的三大需求：高速處理需求、巨量資料處理以及儲存需求。同時，在整個平行運算系統之下，我們也實作了數種人工智慧演算法，例如 SVM（Support Vector Machine）和 LR（Logistic Regression）等，做為策略模擬的子系統。本架構包含下列三種主要的雲端運算技術：

1. 使用 Apache YARN 以整合整體系統資源，使叢集資源運用更具效率。
2. 為滿足高速處理需求，本架構使用 Apache Storm 以便處理海量且即時之資料流。同時，借助該框架，可在數十毫秒之內，運算上千種市場狀態數值供模型建模之用。
3. 運用 Apache Spark，本研究建立了一套分散式運算架構用於模型建模。藉由使用 Spark RDD（Resilient Distributed Datasets），本架構可將 SVM 和 LR 之模型建模時間縮短至數百毫秒之內。

為解決上述需求，本研究設計了一套 n 層分散式架構且整合上列數種技術。另外，在該架構中，我們使用 Apache Kafka 作為整體系統之訊息中介層，並支持系統內各子系統間之非同步訊息溝通。