國立政治大學資訊管理學系

碩士學位論文

指導教授：劉文卿博士、張景堯博士

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

Distributed computing system and big data real-time processing structure—based on YARN, Storm and Spark

研究生：曾柏崴

中華民國一零四年八月
# Table of Contents

Table of Contents ............................................................................................................ i
Table of Figures ........................................................................................................... iii
Table of tables ............................................................................................................... iv
【Abstract】 .................................................................................................................. v

**Introduction** .................................................................................................................. 1

- The Background of the High-Frequency Trading System ........................................ 1
- Design the n-tier Distributed Computing Architecture ........................................ 2

**Related Work** .............................................................................................................. 3

- The N-tier Distributed Computing Architecture .................................................. 3
- High-Throughput Distributed Messaging System——Kafka and Netty ..................... 4
- Real-time Streaming Data Processing——Storm ...................................................... 5
- Forecasting Model Building——Spark ................................................................. 7
- Resource Management——Cloudera and YARN ................................................. 8

**System Architecture** .................................................................................................. 9

- Overview of the Distributed Computing HFT System ........................................ 9
  
  I. Presentation Tier (PT) ........................................................................ 11
  II. Front-end Switching Tier (FST) ....................................................... 11
  III. Back-end Switching Tier (BST) ....................................................... 12
  IV. Real-time Business Logic Tier (RBLT) ............................................ 12
  V. Non-real-time Business Logic Tier (NRBLT) ..................................... 14
  VI. Data Access Tier (DAT) .................................................................. 15

- Lightning Calculation for Market States and Low Latency Storage——State Center ............................................................ 16
- Build Model Fast Over Big Data Sets——Plan Center ................................. 18
- Forecast Market Trends Rapidly and Accurately——Trade Center ............... 19

**Experiments** ................................................................................................................. 21

- The Integration of State Center and Trade Center ......................................... 19
- Cluster Resources Management ........................................................................ 20

- Experiments ............................................................................................................. 21
- Experimental Environment .................................................................................... 21
Table of Figures

Figure 1: A Simple Topology ........................................................................................................6

Figure 2: High Level Architecture of Storm [1] ..........................................................................7

Figure 3: The n-tier distributed architecture of the HFT system ..............................................10

Figure 4: Real-time Business Logic Tier in Detail ......................................................................14

Figure 5: State Center – A Topology .......................................................................................17

Figure 6: State Center – B Topology .......................................................................................18

Figure 7: The Original Design for Trade Center ........................................................................19

Figure 8: The architecture of the HFT topology ........................................................................20

Figure 9: The average cost time of market states computation .................................................23

Figure 10: The average cost time of market states computation ..............................................23

Figure 11: The number of market states are computed for N futures per seconds ....24
Table of Tables

Table 1: The detailed information of the cluster ......................................................... 22

Table 2: The detailed experimental results of the State Center – A ............................... 24
【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 research, 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, RDD), 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.
Keywords: Apache YARN; Apache Storm; Apache Spark; Big Data Processing;

Real-time Forecasting
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 a HFT system, the calculation speed is one of the most critical requirements. A 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 a 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 a HFT system is big data storage. For the enormous amount of data (millions of market state per second for all futures), a 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’s one of the reasons we research the HFT architecture. Eventually, we want to resolve foregoing three main requirements for establishment of a 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 research 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 chapters.

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 chapters.

**RELATED WORK**

**The N-tier Distributed Computing Architecture**

In this research, 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 the research of Manuel and AlGhamdi [10], 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 the paper of Ding, Hu and Sun [11], 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 the paper of Joshi [14], they 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 the paper of Kreps, Narkhede and Rao [13], they 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 [12]. 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! Engineering. Based on the official website, Netty is asynchronous, high-performance, low resource consumption and low latency [15].
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 [16]. 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 administrator to use [1]. 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 [2]. 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 [3].

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/Reduce framework on some specific cases. In the HFT systems, because the Map/Reduce 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 [1]. 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](image)

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) [1]. Generally, the Storm is 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 [1]

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 the paper of Zaharia, Chowdhury, Franklin, Shenker and Stoica [17], Spark is a very useful framework to process large data sets. That research introduces a concept called Resilient Distributed Datasets (RDDs). A 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 [17].

The HFT systems use Spark on machine learning model building over HDFS data. In the paper of Zaharia, Chowdhury, Das, Dave, Ma, Mccauley and Stoica [18], they 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 [18]. 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 cost 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 the paper of Zaharia, T. H. T. D. M., Bayen, Abbeel and Hunter [19], by Spark streaming, they 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 the paper of Buyya, Broberg and Goscinski [20], 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 the users can use it to configure everything they need [20] [21]. 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 the paper of Vavilapalli, Murthy, Douglas, Agarwal, Konar, Evans and Baldeschwieler [22], 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 this paper, YARN provides: greater scalability, higher efficiency, and enable lots of different frameworks to efficiently use the cluster resources [22]. 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.
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 [4] [5]. 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 the users browsing online. For speeding up the loading rate and lowering the latency of access time, the Node.js 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 the users to back-end cluster. All front-end services in PT can’t 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, 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. For efficiency reason, if the request data are stored in MongoDB, the front server accesses the data directly from MongoDB by itself. Then, the request will return and won’t be sent to the back-end Kafka message middleware. This design improves the performance of the HFT system and reduce the user response time.
Otherwise, the FST makes the HFT system more secure. All front requests must to be dispatched to Kafka through front-end switching tier, and not allow being thrown to the back-end cluster for security concern. This tier is important for the response performance of the front-end requests and the security of the HFT system.

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.

The BST contains one service, called Façade. The Façade service will transmit the requests from FST to specific back-end services. And, if we add more nodes (or mini clusters) to the HFT system, we can just let the Façade service send the requests to the specific cluster. Furthermore, we also can add some security functions to the Façade service.

By the design of this Façade service, we have two main advantages:

1. Integrated Security Function: We can monitor all requests from the users that flows through the Façade service. For the security concern, the security relative functions can be added to the Façade service. As a result, the Façade service can identify the user abnormal activities and react to it in real time.

2. Loosely Coupled Architecture: The Façade service makes the HFT system more maintainable. If we want to add some services (or clusters) into the HFT system, we just easily modify some configurations of the Façade service. There is no need to modify other code or program architecture.

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 market states calculation. It contains two important services: **State Center** and **Trade Center**.

The following parts describe the State Center and the Trade Center:

1. State Center: State Center is a Storm topology that processes streams of real-time quotes promptly to calculate millions of market states per second. The 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. The State Center needs to calculate market states in tens of milliseconds to hundreds of milliseconds. That’s really a critical challenge and this research want to propose an architecture to resolve the requirement of fast calculation. As a result, we design the State Center by implementing the Apache Storm framework to calculate market states in time. We will prove the performance of the State Center in the *Experiments* chapter.

2. Trade Center: If Trade Center has running machine learning models for forecast of market trends when State Center sends the market states to it, the running models will calculate signals of buy or sell. The signal will be sent to the Data Access Tier (DAT) and the DAT will send a trading order to outer trading exchange center. Furthermore, the Trade Center has a complicated mechanism to adapt the running models to the real-time trading trend.

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 the Storm topology. The Netty is built in Storm, which helps us with the high-frequency transmission of messages. The Figure 4 describes this tier further.

Figure 4: Real-time Business Logic Tier in Detail

In this tier, the RBLT will send the messages to the Data Access Tier when it needs to communicate with databases and outer services, such as real-time quotes sources, trading exchange center or HBase. The Data Access Tier will be described in the following sections.

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

The 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. The R Academy offers a platform for the 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.

The NRBLT provides two main services:
1. Plan Center: The Plan Center is responsible for building trading strategy models. In the Plan Center, the users can use SVM or LR algorithm to build trading strategy models.

2. R Academy: This service offers an R program development platform for the users. After the creation of the R models, the users can put the strategies on the Trade Center to test them in real time.

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. The DAT also sends several type of K-Bar from different time granularity.

The DAT has three databases to store different type of data:

1. MongoDB: The MongoDB stores some user relative data, such as the user account data, the trading strategies that created by the users and website dynamic content. Most actions of the online users will be responded quickly because the access time of MongoDB is very low.

2. HBase: The State Center saves the market states data in HBase. By HBase, we can write large data in a very short period of time. If the HFT system support 1500 futures calculation, the DAT needs to store 1,977,000 market states per second (1500 * 1318 market states for every second K-Bar) and 35,586,000,000 market states per day (1500 * 1318 market states for every second K-Bar * 18,000 seconds for every day). That’s big data.
3. HDFS: Before the Plan Center builds model, it reads large market states data from HBase and writes them into HDFS. Then, the Plan Center use the data by the Spark RDD to build strategy models.

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.

After our improvement for the State Center, we combine all ComputeStateBolt into a SymbolComputeStateBolt. The SymbolComputeStateBolt is responsible for the all TA calculation of 1 or N futures. We call that “State Center – B”. Figure 6 shows the new topology.
The performance of the “State Center – B” is better than previous “State Center A”. We improve and simplify the messages transmission between the KafkaSpout and the SymbolComputeStateBolt, which reduce a lot of data transmission time. The Experiments chapter shows the performance result of the State Center A and B.

**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 the 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 need 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 [6] [7].

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, the users can choose the investment strategies that they want to use on web pages. The Original architecture of Trade Center shows in Figure 7.

![Figure 7: The Original Design for Trade Center](image)

**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.

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 chapter: Experiments.

![Figure 8: The architecture of the HFT topology](image)

**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 [8].

**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 the HFT system, the State Center computes market states and then the Trade Center uses the market states to forecast the market trends. The whole process must be finished in 1 second because the HFT system forecasts the future trends after 1 second. As a result, the State Center must do all calculation in 500 milliseconds and the Trade Center is the same. Because of the State Center needs to calculate large number of market states, we mainly examine the performance of the State Center. In this section, we present the experimental results to show that the architecture’s performance is good enough for high-frequency trading.

**Experimental Environment**

For evaluating the architecture’s performance, this research 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 research specifically designs experiments for the State Center. We will compare the results with 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 execution environment is shown on table 1.
Table 1: The detailed information of the cluster

<table>
<thead>
<tr>
<th>Host</th>
<th>CPU</th>
<th>RAM</th>
<th>HDD</th>
<th>OS</th>
<th>Storm Role</th>
<th>Zookeeper</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td>nccu-master</td>
<td>Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz</td>
<td>7.7 GB</td>
<td>917.6 GB</td>
<td>Ubuntu 14.04</td>
<td>Storm UI</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>nccu-mgmt</td>
<td>Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz</td>
<td>3.8 GB</td>
<td>458.8 GB</td>
<td>Ubuntu 12.04</td>
<td>Storm Master</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>nccu-n01</td>
<td>Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz</td>
<td>3.8 GB</td>
<td>459.1 GB</td>
<td>Ubuntu 12.04</td>
<td>Storm Supervisor</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>nccu-n02</td>
<td>Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz</td>
<td>3.8 GB</td>
<td>459.1 GB</td>
<td>Ubuntu 12.04</td>
<td>Storm Supervisor</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>nccu-n03</td>
<td>Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz</td>
<td>3.8 GB</td>
<td>459.1 GB</td>
<td>Ubuntu 12.04</td>
<td>Storm Supervisor</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>nccu-n04</td>
<td>Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz</td>
<td>3.8 GB</td>
<td>459.1 GB</td>
<td>Ubuntu 12.04</td>
<td>Storm Supervisor, Follower Broker</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>nccu-n05</td>
<td>Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz</td>
<td>3.8 GB</td>
<td>459.1 GB</td>
<td>Ubuntu 12.04</td>
<td>Storm Supervisor, Leader Broker</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>nccu-n06</td>
<td>Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz</td>
<td>3.8 GB</td>
<td>459.1 GB</td>
<td>Ubuntu 12.04</td>
<td>Storm Supervisor, Follower Broker</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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 cost time of these k-bars and calculate the average value. The Figure 9 shows the performance result of State Center – A, the Figure 10 shows the performance result of State Center – B and the Figure 11 shows the numbers of market states under N futures. The Table 2 shows the experimental comparison between the State Center – A and the State Center – B.
Figure 9: The average cost time of market states computation for different number of futures – State Center A

Figure 10: The average cost time of market states computation for different number of futures – State Center B
**Figure 11:** The number of market states are computed for N futures per seconds

**Table 2:** The detailed experimental results of the State Center – A and the State Center – B

<table>
<thead>
<tr>
<th>Different numbers of Futures (N)</th>
<th>Average cost time of State Center – A (ms)</th>
<th>Average cost time of State Center – B (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>175</td>
<td>59.85</td>
</tr>
<tr>
<td>40</td>
<td>374.73</td>
<td>95.45</td>
</tr>
<tr>
<td>60</td>
<td>588.05</td>
<td>124.32</td>
</tr>
<tr>
<td>80</td>
<td>998.89</td>
<td>169.49</td>
</tr>
<tr>
<td>100</td>
<td>-</td>
<td>222.07</td>
</tr>
<tr>
<td>120</td>
<td>-</td>
<td>233.76</td>
</tr>
<tr>
<td>140</td>
<td>-</td>
<td>299.49</td>
</tr>
<tr>
<td>160</td>
<td>-</td>
<td>396.24</td>
</tr>
<tr>
<td>180</td>
<td>-</td>
<td>514.18</td>
</tr>
</tbody>
</table>
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 need to calculate 65,900 market states for each seconds.

The ExpStateReceiverBolt will compute the average cost time of the market states computation. The cost 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 9, we can see the average cost time of the State Center – A for each k-bar’s market states computation. When the number of futures is 10, which means the HFT system processes 20 futures’ quotes at the same time, the average cost time of market states computation is 175.0 milliseconds in State Center – A. While the number of futures is 40, the average cost time is 374.73 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 seconds (1,000 milliseconds). Theoretically, if we want to support 1,000 futures and don’t delay than 1 seconds, 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.

On the other hand, in Figure 10, in the State Center – B architecture, if the HFT system processes 20 futures’ quotes at the same time, the average cost time of market states computation is 59.85 milliseconds in State Center – B, which is almost one third
of the State Center – A. When 40 futures, the average cost time is 95.45 milliseconds. The State Center – B has better performance than the State Center – A. And, The State Center – B can handle more futures in the same number of nodes (supervisors). After much experiments, we decide to adopt the State Center – B solution.

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 237,240 market states per seconds with 514.183 milliseconds latency. The latency of our architecture is pretty low. In the situation of 6 supervisors and 180 futures, the cost time of all computation of market states is 514.2 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 trends forecasts of **180 futures** with just **6 machines** and don’t 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.
When we design the architecture and develop this system, we face two main obstacles. They will be experiences for other people:

1. Difficult to debug and find issues on distributed computing architecture. Maybe a log centralized management sub-system can be added to the system. The log management system can help us to collect all logs from every services and we can view the logs by the system easily.

2. The techniques in the architecture is too complicated to learn, deploy and maintain.

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.

NOTES
This research is partially supported by TSMC Campus Collaborated Plan and partially supported by the “The Cloud for the Strategy Trading” project with Chunghwa Telecom.

REFERENCES


tzu98/Optimizing%20Hadoop%20for%20the%20cluster. pdf, Retrieved online October.


