Real-time Calculating Over Self-Health Data Using Storm

Jiangyong Cai[1, a], Zhengping Jin[2, b]

[1]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China;

[2]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[a]jy_cai90@163.com, [b]zhpjin@bupt.edu.cn

Keywords: self-health data, real-time calculate, Storm, Kafka, wearable device.

Abstract. With the continuous development and popularity of smart wearable devices, more and more people tend to use the devices to record their health indicators and exercise indicators. Thus a larger amount of indicators called self-health Datais generated all the time. Obviously, it is necessary to process the data in real-time. For example, it may lead to serious problems when someone have an emergency, but if we can process the data in real-time, such situation can be avoidable. However the existing treatments have deficiency in real-time processing. This paper proposed a real-time processing scheme for the self-health data from a variety of wearable devices. We designed a framework using Apache Storm, distributed framework for handling stream data, and making decisions without any delay. Apache Storm is chosen over a traditional distributed framework (such as Hadoop, MapReduce and Mahout) that is good for batch processing. We contrasted different methods to verify the effectiveness of the proposed framework, and we also provided real-time analytic functionality over stream data to show and to improve the efficiency greatly. In our framework we have improved the efficiency by 68 percent compared with the old method of using regular task with DB cluster.

1. Introduction

Big Data that defined as “high-volume, high-velocity and/or high-variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization”[1] by the Gartner, has been widely used in variety of fields such as Big Science, RFID, Internet search indexing, astronomy and so on. Growing number of smart wearable devices like Apple watch or mi-band, are producing large amounts of personal healthy indicators data all the time today. These data we called them self-health big data has become an important basis for people who concerned about their health indicators in real-time. Real-time processing[2], providing the real-time status information of personal Health and sports, is indispensable for the management of ones’ sports and health.

Stream data processing[3] is time sensitive which requires data preprocessing before doing analysis. It converts structured and unstructured, high velocity, high volume Big Data into real-time value continuously. Furthermore, these volumes should be delivered with low latency, and processed with ease, even when the velocity is high. Recently we have seen several Big Data frameworks (such as Hadoop[4], Map Reduce[5], HBase[4], Mahout[6], Google Bigtable[7], etc.) that address the scalability issue. They are good at batch based processing. But in our application scenarios, we need process the stream data in real-time. While the real-time system demands the streaming data processing. In order to realize the real-time stream data processing, we choose Apache Storm. It is a distributed framework offering streaming integration with time-based in-memory analytics from the live machine data as they coming in stream. It generates real-time, low latency results and has an easy cluster setup procedure. It also has no hidden obstacles.

In this paper, we have the following contributions:

We have developed a real-time framework. As a broker, we use Kafka[8] to collect the data from MongoDB[9] and continuously transfer them to Apache Storm[3]. Inside Storm, we apply two calculating methods to analyze the data. We design two different kinds of bolts, one kind is to do some pretreatment and the other kind is to process the intermediate results.

We have experimentally shown that our framework calculate self-health data without sacrificing accuracy. The results turned to be nearly the same as the old method. It also has a low latency to analyze data. In our framework, we can process the data in real-time while the old method can't. We have also improved the processing efficiency by about 68 percent compared with the old method. The calculating efficiency is about 2000 per second with our framework while the old method is only about 1180. With our framework we have reduced the data processing time.

2. Background

2.1 Distributed DB Cluster

We need a high-performance database to store the big self-health data coming from the intelligent device. We choose high scalability database called NoSQL. Scholars have indicated that it exhibits more advantages compared with RDBMS. MongoDB is a representative NoSQL database. We can build a Cluster to improve the efficiency and scalability.

2.2 Self-Health Stream Data

We continuously collect self-health data into our database cluster and gather the data we called stream data from our database cluster. In order to analyze the different indicators, we need to use different criteria, so we use different items to mark different indicators. Each indicator has its own normal range.

2.3 Real-time Issues and Apache Storm

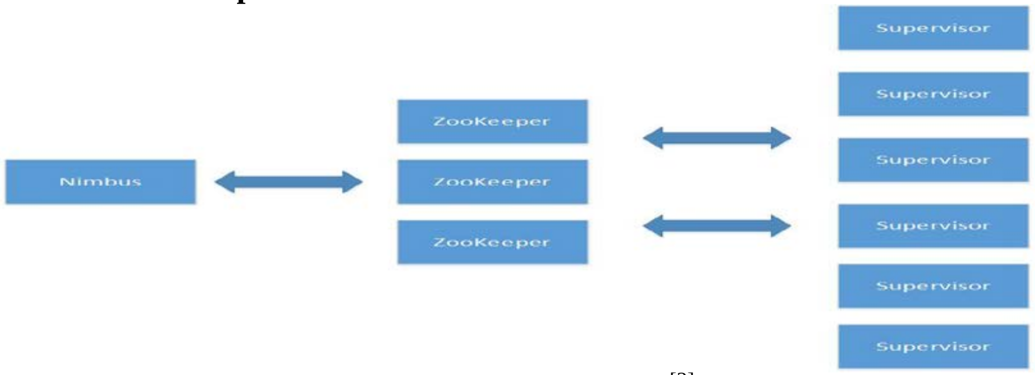


Fig. 1 Strom Architecture[3]

We require a scalable distributed analyzing environment. To solve the issue, we may use a distributed solution like Mahout, Hadoop, etc. But they can't handle the real-time data. Storm is a distributed real-time computing system[9] developed by BackType and open sourced by Twitter in 2011. Its main application scenario is real-time analysis, online machine learning, continuous computing, distributed RPC, ETL, etc. Storm has a simple architecture (in Fig 1) with three sets of nodes:

1) ZooKeeper: Communicates and coordinates the Storm cluster.
2) Nimbus: Master node distributes jobs and launches workers across the cluster.
3) Supervisor: Communicates with Nimbus through Zookeeper.
4) Topology: Storm consists of Topology, Spout and Bolt (in Fig 2)

Spout is the input stream source which can read from external data source. Bolts are processor units which can process any number of streams and produce output streams. Topology is complex multi-stage stream computation and a network of Spouts and Bolts. In a Storm cluster, the master node coordinates and communicates with all the worker nodes.

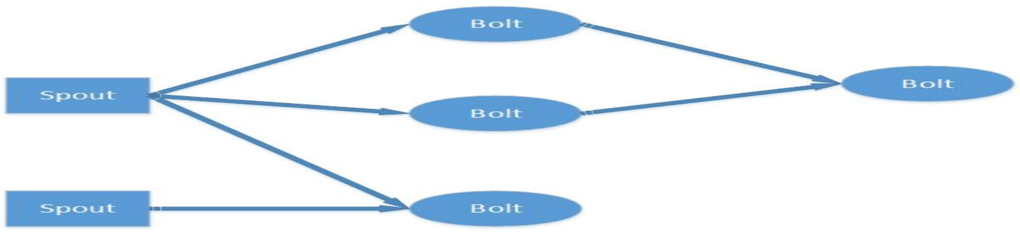


Fig. 2 Storm Topology[2]

3. Real-time Framework Based on Storm

Our real-time calculating framework consists of data preprocessing, sports and health prediction, and real-time health notifications. The streaming data comes in raw format. This format data should be processed and be analyzed. After processing, it is converted to multidimensional points, such as the heart rate point or respiratory rate. And then, we use different methods to analyze and produce the analyzing results. Finally, we compare the results with different normal values and the historical value to produce the final sports and health advice.

Our real-time framework is divided into five parts. They are as follows:

3.1 Message Broker

We are continuously collecting the self-health data of each wearable device. We put these data into a unified format and store them into the cluster of MongoDB using the data services interfaces based on REST[10] architecture. We read those data from the MongoDB cluster and transport them through Message broker. Integrating the message broker to Storm is a challenge. There are several message brokers (e.g. RabbitMQ[11], Apache Kafka etc.) available to integrate with Storm. We have concerned about RabbitMQ. It is a practical realization of Advanced Message Queuing Protocol (AMQP)[12]. At last, we choose Apache Kafka because it has nearly the same property of RabbitMQ. It is version compatible and widely in combination used with Apache Storm and it is stable. Kafka creates a dedicated queue for transporting message. It also supports multiple source. So Kafka ships (in Fig 3) those self-health data to Storm's input source.

3.2 Stream Data Mining Module

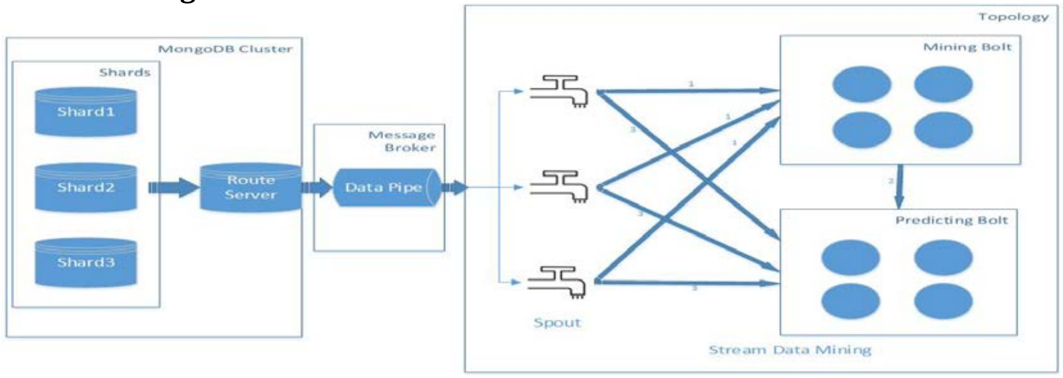


Fig. 3 Stream Data Processing Using Storm

We have implemented a clustered network using Storm as in Fig 3. The whole clustered network is called Topology. Storm has input sources called Spout, it also has working nodes called Bolt. The Spout receives the self-health data as tuples through Apache Kafka. Kafka ensures guaranteed tuples delivery. It transports data to the Spout. Spout emits those tuples to Bolts. We have written several Mining Bolt for the preliminary analyses and some Predicting Bolt to predict health status.

3.3 Preliminary Analysis and Predicting

Firstly, the Apache Kafka gathers the real-time data into the Spout continuously. And then the Spout delivers the data to different Mining Bolts. Then, we use the Calculation Method to calculate the data and change them to the preliminary analysis results and store them into the DB cluster. Secondly, the Spout delivers the preliminary analysis results to different Predicting Bolts. We will use a method to compare the results with the historical data and the normal range about the health indicators to generate deeply analysis results. Finally, we will storage the results into the DB cluster.

3.4 Generate Health Notifications

When we discovery new finally results in the MongoDB cluster, we will analyze the results and decide whether it is necessary to generate advices to the user or not. If needed, we judge the exceptions are accidental or regular and give some health suggestions. Then, we will send the analysis result and the advice to the user by SMS or wechat message, we also showed this result and advice on our portal website.

3.5 Scalability

The framework is generic. We can build a message broker cluster using Kafka. It is scalable and highly available. We also built a DB cluster, the MongoDB cluster, which is also available and scalable. This means when we need to store more data or expand storage capacity, we can add more shards[9] into the cluster. Meanwhile, we can add executors or workers in Storm's Spout and Bolt to increase parallelism. In this way, we can accommodate these large volumes of data.

4. Experiment and Result

4.1 Experiment Environment

Our Storm cluster consists of 3 servers. Our DB cluster also contains 3 nodes. Each of the servers and DB nodes has Intel(R) CPU processor, single NIC card, 512 GB hard disk and 2 GB DDR3 RAM. We have installed Linux Centos v6.4 64 bit OS in each of the servers and DB nodes along with the JDK/JRE v 1.7. We have installed Apache ZooKeeper 3.4.6 with Apache Strom version 0.9.4-rc2.In each of the DB nodes, we have installed the MongoDB v 2.6.3 64 bit database system.

4.2 Testing Data

We have presented the accuracy of the framework in this part. Table 1 shows the Apache Storm setup. We run our experiment on a Storm cluster. It has 4 parallel Spouts, 4 Mining Bolts and 4 parallel Predicting Bolts as Table 1 shows.

Table 1 Storm Topology

Component	The Number of Parallelism
Spout	4
Mining Bolt	4
Predicting Bolt	4

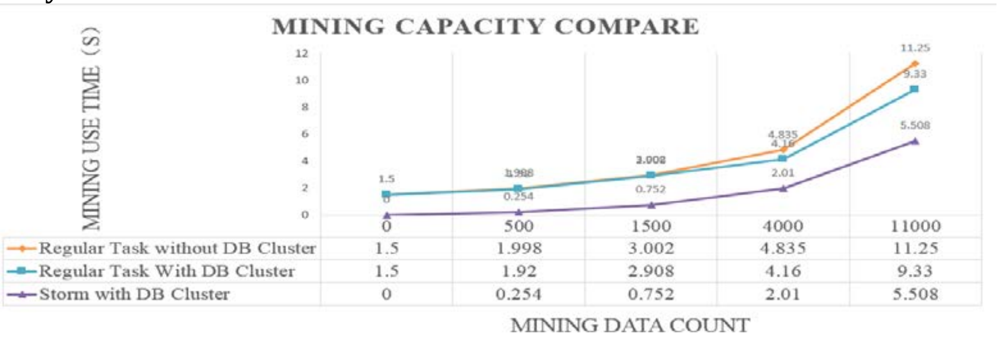
Table 2 Testing Data

Testing Count	Mining Result Count
500	78
1500	202
4000	320
11000	6054

To test the ability of data processing of the platform, we designed three methods. Firstly, we choose the way of regular task to simulate real-time processing. The first method is regular task with MongoDB but not MongoDB Cluster. The second method is regular task with MongoDB Cluster and the third method is Apache Storm real-time platform with MongoDB Cluster. Secondly, we choose different count of testing count to test the three methods. And last, we compare the results and produce a conclusion.

Table 2 shows the testing data we choose. We selected four different number of test data, they were 500, 1500, 4000 and 11000, and the mining result count are 78, 202, 320 and 605. We use the fore test cases to test the data mining methods to check the capacity.

4.3 Result analysis



	0	500	1500	4000	11000
Regular Task without DB Cluster	1.5	1.998	3.002	4.835	11.25
Regular Task With DB Cluster	1.5	1.92	2.908	4.16	9.33
Storm with DB Cluster	0	0.254	0.752	2.01	5.508

Fig. 4 Data Mining Results Contrast

Fig 4 shows the comparison of the three methods about the capacity of the testing data. Just as the graph shows, the efficiency of the first method is the lowest, and the second method is a little higher than the first one, but they are very similar. They all need some time to process some initial work when start a task. The capacity of the real-time platform is the highest and there is no need to process initial work because it is running all the time. The efficiency is far higher than the first two methods.

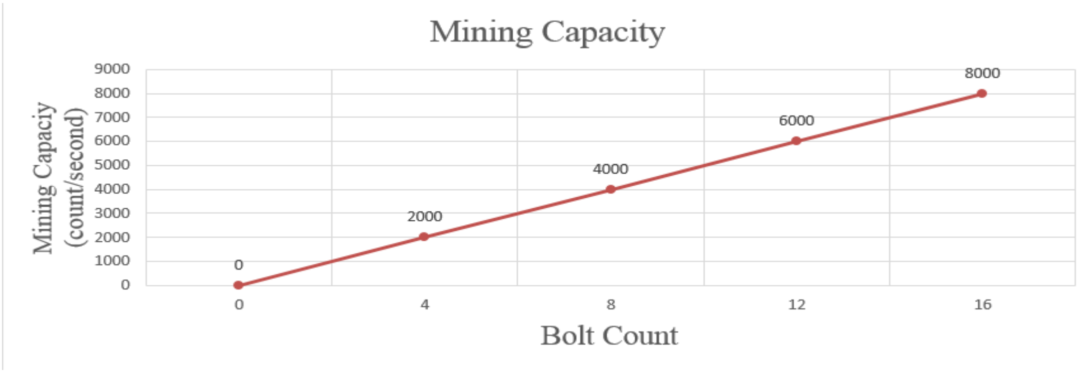


Fig. 5 Platform Data Mining Capacity

And then, we did some other test. Each time, we define different Bolt counts to test the efficiency. The result is as the Fig 5 shows. If we add the Bolt count within the hardware processing capability, we can increase the efficiency and the variety is nearly linear.

In view of the above facts, we decided to use Apache Storm. It is scalable and it has ensured data processing, and finally it is perfectly suited to process the stream data.

5. Conclusion

In this paper we have implemented a real-time framework using Apache Storm. Storm guarantees data delivery and has the ability to manage the stream data. Furthermore, integrating Storm with Kafka makes the real-time framework more generic to fit with other environments. We envision our real-time framework to be the key part of our Personal Health Service Platform. We can integrate it to other systems like monitoring system log to measure the system's operational performance.

Acknowledgements

This work is supported by NSFC (Grant Nos. 61300181, 61502044), the Fundamental Research Funds for the Central Universities (Grant No. 2015RC23).

References

[1] Beyer M A, Laney D. The Importance of 'Big Data': A Definition. Stamford, CT:Gartner, Jun.2012.

[2] Wenjie Y, Xingang L and Lan Z. Big Data Real-Time Processing Based on Storm. Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on IEEE, 2013: 1784-1787.

[3] Solaimani M, Khan L, Thuraisingham B. Real-time Anomaly Detection over VMware Performance Data Using Storm. Information Reuse and Integration, 2014 IEEE 15th International Conference on. IEEE, 2014: 458-465.

[4] Information on:http://hadoop.apache.org/

[5] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 2008, 51(1): 107-113.

[6] Anil R, Dunning T, Friedman E. Mahout in Action. Greenwich, Conn Manning London:

Pearson Education, 2012.

[7] Chang F, Dean J, Ghemawat S, et al. Bigtable: A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems, 2008, 26(2): 4-26.

[8] Information on:http://kafka.apache.org/

[9] MongoDB, [Online]. Available: https://www.mongodb.org/

[10] Tim O'Reilly. Rest vs Soap at Amazon. [Online]. Avaialble: http://www.oreillynet.com/pub/wlg/3005, 2003, 15: 724-727.

[11] Information on:https://www.rabbitmq.com/

[12] Xiong X, Fu J. Active Status Certificate Publish and Subscribe Based on AMQP. 2011 International Conference on Couputational and Information Sciences, Oct. 2011: 725-728.