SpeedStream: A Real-Time Stream Data Processing Platform in The Cloud

Li Zhao, Zhang Chuang, Xu Ke-fu*
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China
E-mail: zhangchuang@iie.ac.cn

***Abstract*—SpeedStream is a universal distributed platform that can handle with massive data flows with the features of low coupling, high availability, low latency and high scalability. Focusing on the core technologies of real-time stream computing platform in cloud environment, this paper conducts a series of researches and implementation of the system. First of all, aiming at the availability of real-time streaming computing platform, we design a high availability framework based on Zookeeper. It ensures fault detection and recovery of process level and node level timely by monitoring heartbreak of each modules and strategy of fault migration. Secondly, in order to increase the application types of the platform, by means of directed cycle detection and iteration protection, we design a real-time streaming computing model that based on directed graph with sources and sinks, which can not only satisfy the needs of common DAG computing services, but also support iteration computing services including directed cycle, bidirectional arcs and annular arcs. In addition, the platform can realize personalized task scheduling strategy for users by establishing task allocation matrix and optimize task allocation model. Finally, in order to solve the many-to-many dynamic load-balancing between tasks, we apply scheduler with status and distributed session table. It overcomes the difficulty of maintaining consistency of session without global session table. We also testified the convergence of this method. The experiment indicates that the throughput and data processing delay of SpeedStream are superior to other alternatives in dealing with the businesses of iteration applications, high traffic fluctuation applications, and high demand of load-balancing applications. This platform provides reliable, universal, and real-time solutions to process massive data flows, such as to process the real-time trading data in e-commerce, to analyze sensing flow in internet of things, and monitor traffics of the Internet.**

Keywords—real-time stream processing; computing model; iteration computing services; task allocation matrix; dynamic load-balancing

I. INTRODUCTION

The epoch of big data has walked in, which causes the proportion of massive stream data increased gradually. Stream data is a sequence of ordinal, massive, rapid and continuous data. In general, it can be regarded as a dynamic data set that can infinitely increase with the continuation of time[1]. Stream data has three characteristics: (1) Immediacy: the real-time data element of stream data requires instant response and disposition. (2) Randomness: system is unable to control the sequence of the new-arrival data element, no matter these data elements are in one data flows or across multiple data flows. (3) Infinity: the potential size of data flows may be innumerable. (4) Transient: once the data was processed, unless preserved specifically, otherwise it cannot be taken out to deal with again. Or the costs of retrieve the data would be expensive[2].

The real-time stream data processing and mining has a capacious prospect, for instance, the real-time delivery of messages in social networks, the real-time monitoring of network traffic, the rapid analysis of sensing flow data in internet of things, the real-time processing of various data flows on the Internet and so on[3][4]. Therefore, the modeling and handling of stream data has obtained extensive research. However, the size of stream data grows as geometrical progression, which presents a new challenge in the processing model and platforms of stream data. At the same time, the rapid development of cloud computing provides us with massive computing resources and storage capacity, as well as the common services and high availability. Consequently, researchers are turning their focus to processing stream data in cloud environment, which is meant to providing common and reliable processing services.

However, traditional cloud computing platforms are usually batch processing platforms (such as Hadoop). The input of batch processing job is the pre-stored static data, and jobs with predictable volume would be ceased when the data processing finished[5][6]. Different from batch processing, the input of stream data processing is unremitting and fluctuant in data size. Therefore, it is necessary to find new solution to conduct streaming computing in cloud environment.

Aurora is one of the classic representative of stream data processing platform in early time[7].The original intention of designing Aurora was to overcome the shortcoming of database when dealing with data stream. Moreover, Aurora firstly used the arrow diagram model to define data flows, which provides some basic data stream operation to users. Through the interaction of scheduler module, QoS monitoring module, and load shedder uninstall module, Aurora achieved the data stream computing.

Borealis made some improvement on the basis of Aurora[8]. It proposes a technology of dynamic modifying result, which was beneficial for users to modify the results into the one that can suit their requirement. For example, it could remove the stream data that cannot meet the filter conditions.

Hadoop Online is a stream processing data platform depending on Hadoop platform[9]. The input of Map was stored in cache by HOP, and then used a thread to send the contents of the cache to Reducer with the pipe connection for processing periodically. The intermediate data between modules is connected by pipes. Hadoop Online maintains the programming

Supported by the "Strategic Priority Research Program" of the Chinese Academy of Sciences, Grant No. XDA06031000

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

interfaces and fault tolerance model of the traditional MapReduce framework.

Storm is a stream computing platform that is widely used[10]. There are two basic components in the job that defined by Storm: Spout and Bolt. Spout is the source of a data stream, in which users can freely define the source of data stream. Bolt can handle multiple data streams, and continue to send the data streams to other Bolts after the processing is done. It can achieve many functions, for instance, tuples filter, stream aggregation, flows grouping, and the storage of results into database and so on.

TimeStream is a distributed stream computing platform developed by Microsoft[11]. It is based on Microsoft StreamInsight system. Through DAG computing model, it combines MapReduce batch processing with stream data processing. It provide the functions like dependency tracking, dynamic reconfiguration, and failure recovery and so on.

Spark is a distributed platform developed by Berkeley AMP lab[12]. Spark has the advantages of Hadoop MapReduce, but different from MapReduce, the intermediate output from Job can be saved in cache, and there is no need to read HDFS. The principle of Spark is to divide streams into small time-slice data, and then use batch processing to deal with these data.

The distributed stream processing system S4 of Yahoo! is a common and open source solution to process stream data[13]. S4 has a symmetrical structure, which make all nodes in the cluster are identical. The absence of control center node makes the framework of S4 more simplified. S4 neither provides dynamic load-balancing, nor resource changing during runtime.

DStream proposes a discrete data stream model[14], in which the core concept is to regard a stream computing in a small period of time as a series of batch processings. The well-defined consistency and the failure recovery mechanism by using batch processing system are the advantages of DStream model.

This paper would introduce the real-time stream processing platform SpeedStream designed in the cloud environment. SpeedStream is a distributed platform to deal with massive stream data with characters of low coupling, high availability, low latency and high scalability. It has many functions, including dynamic libraries and executable files interface, dynamic job update and reconfiguration while running, dynamic load-balancing strategy, task migration from failure nodes, various task allocation strategies, and rich UI monitoring information. Users can get rid of negative effects like communication realization, cluster building, cluster maintenance and so on through this platform. So the attention can be focused on the realization of business to shorten the business development period and decrease development and maintenance costs.

The main contributions of SpeedStream are summarize as follows: 1) SpeedStream develops a low coupling computing model based on directed graph with sources and sinks, which can not only satisfy the needs of common DAG computing services, but also support iteration computing services including directed cycle, bidirectional arcs and annular arc. 2) We design a high availability framework, which can timely detect the fault of process level and node level. We also design a failure recovery mechanism. SpeedStream can also provide alarm in UI for users and manual modulation interface after a failure is resolved, thus to make sure that any node failures won't affect business operations and platform services. 3) We present a task schedule model based on task allocation matrix. The platform can realize personalized task scheduling strategy for users by establishing task allocation matrix and optimize task allocation model. 4) In order to solve the issues of many-to-many dynamic load-balancing between tasks, we apply scheduler with status technology and distributed session table. It overcomes the difficulty of maintaining consistency of session without global session table. We also testified the convergence of this method. It can balance the traffic and improve the utilization of resources.

Following that, we would discuss the framework and functional characteristics of SpeedStream platform in detail. In the Section II, we will introduce the current related work. And in the Section III, we will introduce the functions and framework of SpeedStream platform. In the following part, Section IV, we will introduce computing model of SpeedStream, state model of worker, task scheduling model, dynamic load-balancing and design of high availability framework in detail. The Section V focuses on experiment. Finally, in the Section VI, this paper would make a comprehensive summary.

II. RELATED WORK

In the area of stream computing in cloud environment, there already exist some excellent platforms. Inspired by these work, the SpeedStream platform we proposed has made some innovation and optimization. Next, we will discuss the difference between the previous work and our work.

A. Programming Model

The majority of stream processing platform is data-driven computing platform, whose business functions can be achieved by implementing the provided programming interfaces. These platforms adopt traditional batch processing method MapReduce, which is based on DAG model. In this model, users must use the interface specified to set inputting, outputting, processing logic, grouping strategies and so on. For an instance, the inputting of Storm platform must be a group of Tuple sequence, and this precondition cannot be meet by all businesses. Besides, DAG model requires that there cannot be a loop in the computational logic, which means that DAG model is not suitable for the iterative business. As a comparison, The Stream computing model proposed in our paper has two advantages: Firstly, SpeedStream can support not only DAG computing services, but also iterative and feedback services with directed cycle, bidirectional arc and ring arc. In a word, SpeedStream significantly broadening the scope of stream computing model. Secondly, in order to ensure that our work can support for complex business, SpeedStream implemented the function of directed cycle detection and iterative protection. In other words, SpeedStream can automatically extract the loop from job topology, and set the conditions for convergence of iterative calculations, to ensure that business can be successfully achieved.

B. High Availability

For large distributed cluster, a high-availability design is always an important part, because distributed platforms need to

ensure cluster can still work when a node breakdown. For example, in order to protect the safety of tasks, Storm platform monitor the heartbeat of worker nodes, migrate tasks from failure nodes to other nodes [10].However, once Nimbus node breakdown, the function responsible by Nimbus will be unavailable, although the running job and tasks are still running normally. That is to say, a new job could not be submitted to the platform, the cluster state cannot be monitored, and the task cannot be reallocated when node breakdown. In this paper, we adopt multi master node mutual-backup approach based on distributed lock. In this approach, a plurality of master node with non-state and peer structure are compete for distribution lock, the master node getting the lock works. When the working master node fails, it will release the distribution lock, and then other master node which obtains the lock will running instead. So, as long as there is one master node exists, the entire cluster can work normally. In a word, this approach eliminates the master node dependency.

C. *Task Scheduling*

The scheduling strategy of most current platform is simple, only the average allocation strategy and minimum resources occupancy strategy. The former allocate the task to all node averagely; the latter occupies all computing resources of a node in the priority. These two simple scheduling strategies cannot meet the more complex business requirements, for example, as high-throughput business is very sensitive to network bandwidth usage rate, such business should ensure the traffic of each node relatively equal. Therefore, we designed a task scheduling model based on task allocation matrix, based on this model, the user can achieve scheduling strategies personalized to meet the diverse business requirements.

D. *Dynamic load balancing*

In distributed stream computing platform, there is a case that the upstream program cannot know the number of tasks running on a machine downstream when data is sent from upstream to downstream. Under this circumstance, the nodes which already running a multi-task will afford much more traffic and much larger delays than the nodes which runs only one task. Currently, Storm[10],S4[13],TimeStream[11]provide many task grouping strategies, these platforms even can write hashing algorithm automatically to solve the problem of uneven flow of packets. However, as the stream data is time-varying and unpredictable, hashing algorithm is difficult to solve the problem of dynamic load balancing. Moreover, once the task migrates, the hashing algorithm is no more suitable. This paper presents a dynamic load balancing algorithm, this algorithm can make loading and stream grouping dynamically, while ensure that the same packet data can be sent to the same node to process. Even the data are from different nodes, the algorithm can maintain packets consistency in load migration process.

III. PLATFORM ARCHITECTURE

The physical structure of platform architecture and deployment is shown in Fig.1, which includes several Master nodes, Supervisor nodes, Zookeeper nodes and Client nodes, with different modules running on different physical resources, where:

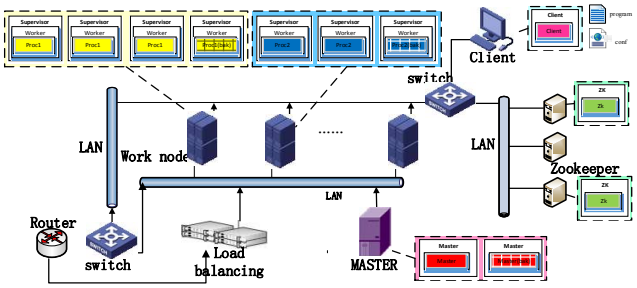


Fig. 1. SpeedStream architecture

1) Zookeeper is an open source system[15][16] which is widely used in distributed clusters to provide reliable distributed coordination services, distributed storage services, configuration and maintenance services, name services and so on. SpeedStream uses Zookeeper as a highly available state storage system for saving working conditions and heartbeat information of other modules.

2) Supervisor running on work nodes is primarily responsible for monitoring Workers assigned to work nodes, starting/closing Workers when necessary, and listen to the update situation of executable programs stored on Zookeeper or dynamic library files. If programs update, it will then download the update them to the local catalogs. Working status of nodes and business processes running in Worker will be passed to corresponding Znode in Zookeeper in the form of heartbeat information.

3) When Supervisor is started, several Workers will be started at the same time. Worker is responsible for starting, stopping or restarting business processes. Worker periodically monitor the status in Zookeeper to perform starting, stopping and restarting job of business processes.

4) Business programs running in Worker is called tasks which is a parallelism of a business process. Programs provided to the platform by way of dynamic link library files or executable files. Executable files needs to read the downstream IP and ports provided by the platform to realize communication by itself.

5) Master nodes compete distributed locks to work. Those master nodes getting locks execute cluster monitoring work. The rest master nodes continued to compete for locks. Masters with locks are responsible for monitoring the heartbeat information of all work nodes and business processes on Zookeeper. When heartbeat timeout, the node can be judged as failure. This will trigger re-allocation of tasks and task migration of failed nodes.

6) Client program running in client nodes provides user interfaces for users to define jobs, set the degree of parallelisms, submit programs, etc. Client terminal would analyze and calculate the above information submitted by users to obtain an allocation strategy of resources and tasks, and upload tasks assigned to each Supervisor and Worker to the corresponding Znode in Zookeeper. When Supervisor and Worker are monitoring Zookeeper, it will download task information and perform these tasks.

IV. KEY TECHNOLOGIES

This chapter will introduce the key technologies of SpeedStream, which including computing model, the Worker state model, task scheduling model, dynamic load balancing and high availability design.

A. Computing Model

This section will introduce our computing model and directed circle detection methods presented in this paper.

1) Model Definition

The computing model based on the directed graph with sources and sinks can be abstracted as follows[17].

Job = (V, S, D, A, W, C)

V：{ v | v ∈V (Job) }

S：{ v | v ∈V, and d-(v) = 0} Φ⊂S⊂V

D：{ v | v ∈V, and d+(v) = 0} Φ⊂D⊂V

A：{ a | a ∈((V-D)×(V-S)), and ∃ (vi∈V-D∧vj∈V-D) makes the directed edge (vi , vj) exist }

W：{ w | w ∈W (Job)}

C：C=(W∪V, E), E={ e | e ∈(W×V), and ∃ (wi∈W∧vj∈V) makes edge (wi , vj) exist }

Among them, V represents the set of vertex in the directed graph with sources and sinks, which can be seen as the process of stream business; S represents the set of vertex whose in-degree is zero, which can be seen as data source; D represents the set of vertex whose out-degree is zero, which can be seen as data terminal; A represents the set of the directed curve that is from one vertex to another vertex, which can be seen as the data flow path. The rank of Job is denoted as m=|V|, and the adjacency matrix of the graph can be expressed as:

$$\begin{bmatrix} a_{00} & \cdots & a_{0m} \\ \vdots & \ddots & \vdots \\ a_{m0} & \cdots & a_{mm} \end{bmatrix} \quad a_{ij} = \begin{cases} 1 & \text{directed edge}(vi, vj)exist \\ 0 & \text{others} \end{cases}$$

W represents the set of Worker which is required in the job; C is a bipartite graph, which represents the condition restriction of executable V (stream processing program) in the Worker, which can be seen as the mapping relations that is between Worker and vertex V.

Fig. 2 is an example of the model. It includes four vertexes V0 , V1 , V2 , V3, which can be seen as four stream processing programs. S, data source of stream processing business, is V0; D, data terminal of stream processing business, is V3; A, the directed curve, respectively is (V0,V1),(V0,V2),(V1,V3) and (V2,V3); the adjacency matrix is $\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$; the set of Worker applied is W0,W1,W2 and W3(each of which contains N Workers); The condition restriction of executable V (stream processing program) is as followed. W0 can execute the business process represented by V0, W1 W2 W3 can execute the business process represented by V1 V2 V3 in a similar fashion.

User defines jobs by the interface, establishes the mapping relation between task and Worker, and distributes the program to the corresponding node, then Worker starts the business procedures and establishes the data connection of upstream and downstream. Data is processed by Job in the following figure.

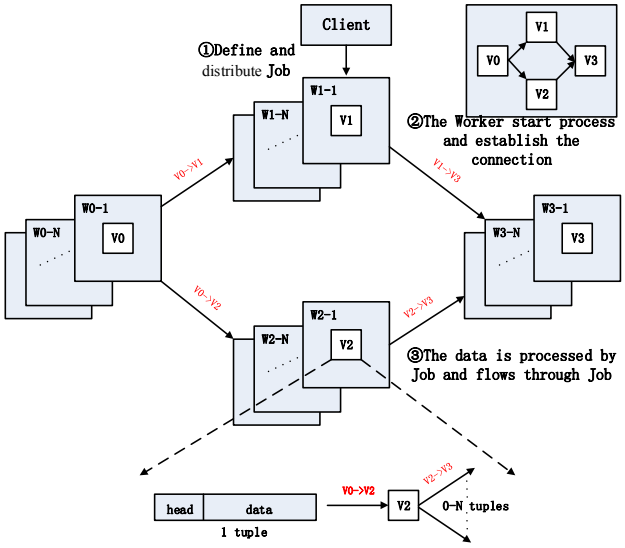


Fig. 2. Example of the computing model

2) Directed Circle Detection

The feature of iterative feedback business on directed graph with sources and sinks is the existence of loops. In order to meet such business needs, we apply directed circle detection algorithm to automatically identify loops. Once a loop is detected, it will extract the loop and set a counter for each packet flowing through the loop. When a packet flows through the loop, the counter will be incremented by 1. Users can set the maximum times of iterations by themselves. Once the counter of a packet exceeds this limit, namely, failing to meet the convergence conditions within the limited times of iterations, the packet will be discarded. This paper uses depth-first traversal method to find all directed circles on the graph. Starting from the source point, traverse all the downstream nodes and push them into a stack. During the process, if one of the nodes is already in the stack, it means the discovery of a directed circle. If there is no directed circle along this path, then return to the parent node for recursive lookup. Algorithm pseudo-code is as follows:

```
Input: source v
Output: directed circle
1:  function: findCycle(v):
2:   ArrayList<Integer> trace;//trace from source
3:   if visited[v]==1
4:     j←trace.indexOf(v);
5:     if j!=-1
6:       hasCycle←true;
7:       while j<trace.size()
8:         output trace.get(j);//output directed graph
9:         j++;
10:      end while
11:    end if
12:  end if
13:  visited[v] ←1;
14:  trace.add(v);
15:  for i=0;i<size(nodes);i++
16:    if e[v][i]==1
17:      findCycle(i);//find the next level
18:    end if
19:  end for
20:  trace.remove(trace.size()-1);//trace back
21:  end function
```

B. Worker State Model

Worker execute tasks with the following functions :1) starting task function (handle_local _tasks); 2) migrating task function (change_ local _tasks) 3) killing task function(exit _local _tasks). Platform can control these tasks by these functions. But what really trigger these actions is the Worker's state-transition of Worker state model. Worker State model are defined as follows:

$$M = (P, Q, \Sigma, \Gamma, \delta, \varepsilon, S, F), \quad \delta(p, q, \sigma) = \begin{cases} \delta(p, q, \sigma) & p|q \in F \\ p|q & p|q \in P \times Q - F \end{cases}$$

P：worker's state set in Zookeeper,{STAT_VOID, STAT_STANDBY, STAT_LIVE_ING }

Q：worker's local state set,{worker_waiting, worker_running}

Σ：event which is initiated by the client, {submit_topology, migrate_topology, re-submit_topology, kill_topology}

Γ：action which is generated by Worker, {KEEP_STATUS, TASK_GONE, NEW_TASK, CODE_CHANGED, TASK_CHANGED}

δ：active state transition function which is triggered by events, δ：$P \times Q \times \Sigma \rightarrow P \times Q$.

ε：passive state transition function which is triggered by action, ε：$P \times Q \times \Gamma \rightarrow P \times Q$.

S：initial state, {STAT_VOID|worker_waiting}

F：steady state, { STAT_VOID|worker_waiting ,STAT_LIVE_ING|worker_running}

Worker state model is an eight-tuple model. P represents worker's state set in Zookeeper. STAT_VOID indicates that Zookeeper has no task assigned to Worker. STAT_STANDBY indicates that tasks are uploaded to Zookeeper. STAT_LIVE_ING indicates that tasks in Zookeeper have been taken by Worker. Q represents worker's local state set. Worker_waiting indicates that there are not tasks executing in Worker. Worker_running indicates that there are tasks executing in Worker currently. Σ represents the event which is initiated by the client, such as submit_topology, migrate_topology, re-submit_topology and kill_topology. Γ represents the action generated by Worker, such as to keep state executing currently (KEEP_STATUS), to kill the tasks (TASK_GONE), to execute new tasks (NEW_TASK), to change the program (CODE_CHANGED) and task immigration (TASK_CHANGED). δ indicates active state transition function which is triggered by events. Executing active state function will transfer the worker state. ε indicates passive state transition function which is triggered by action. Executing passive state function will transfer the worker state. S indicates the initial state (STAT_VOID|worker_waiting), Zookeeper has no task assigned to Worker, and there are not task executing in Worker when platform starts. F indicates the steady state. STAT_VOID|worker_waiting indicates that Zookeeper has no task assigned to Worker, and there are not task executing in Worker. STAT_LIVE_ING|worker_running indicates that tasks in Zookeeper have been taken by Worker and executed by Worker. All other states are unstable intermediate state. The worker state model is illustrated in Fig. 3.

After Worker starts, it will send heartbeat to relevant Znode of Zookeeper, while checking it's state in Zookeeper to judge the next action or event. Worker can be in two steady states or four intermediate states. When Worker in steady states, indicating that Worker's local state is synchronize with Worker's state in Zookeeper. When Worker in intermediate states, indicating that Worker's local state isn't synchronize with Worker's state in Zookeeper, and need to achieve a specified steady state. In fig. 3, steady states are those with striped background. Intermediate states are those with white background. The Worker state model is a core part of the platform to run business program and drive all Worker's to work in the SpeedStream platform.

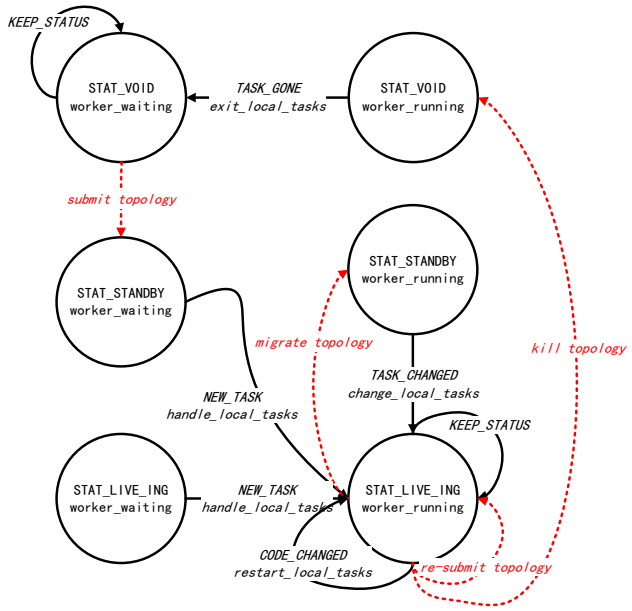


Fig. 3. Worker state transfer

C. Task Scheduling Model

Task scheduling is a hot spot in the cloud computing and distributed computing field. Task scheduling can influence the resource usage rate and computing efficiency of distributed platform. A good scheduling algorithm should comprehensively consider the computing load, internal memory load and traffic load of computing resource, thus allocating tasks to computing units with fewer loads as much as possible to increase the computing efficiency and resource usage rate. It should even allocate two tasks that need communication with each other to the same computing unit to reduce the communication traffic in the network[18][19]. However, in practical usage, it is not easy to find the scheduling algorithm that could meet above conditions at the same time. Besides, application programs of different types usually have different demand on task scheduling algorithms. For example, computing-intensive tasks usually hope that tasks could be more evenly distributed to the computing resources to ensure the usage rate of computing resources so that the nodes would not be the bottle neck due to

lack of computing resources; furthermore, some applications with high throughput are very sensitive to the usage rate of network bandwidth. For such applications, the traffic at each node shall be maintained at an even level as much as possible in the task scheduling process.

It is difficult to satisfy the demand from different types of applications at the same time. Therefore, SpeedStream designs a set of task scheduling model[20] and offers frequently used basic algorithms based on this model. Meanwhile, SpeedStream platform also reserves the interface of the task scheduling algorithm, so the scheduling model we designed could help users realize the scheduling algorithm meeting their own business demand.

The purpose of task scheduling is to allocate m Tasks to n Workers for execution according to the allocation matrix. Therefore, the allocation algorithm could be abstracted to an allocation matrix which matches its own allocation algorithm. Use the allocation matrix to multiply the m×1-dimension Task matrix and the m Tasks would be mapped to the corresponding n Workers. Assume that Task allocation matrix is A, then A is a n×m-dimension matrix:

$$\begin{bmatrix} a_{00} & \cdots & a_{0m} \\ \vdots & \ddots & \vdots \\ a_{n0} & \cdots & a_{nm} \end{bmatrix} \quad a_{ij} = \begin{cases} 1 & \text{task[j] } \textit{is assigned to} \text{ worker[i]} \\ 0 & \text{else} \end{cases}$$

Different allocation matrices could match different allocation strategies, such as even allocation and minimum resource-occupying allocation. Taking even allocation for example, when n<m, even allocation's Task allocation matrix A is:

$$\begin{bmatrix} 1 & & & 1 & & & 1 & & \\ & 1 & & & 1 & & & 1 & \\ & & \ddots & & & \ddots & \cdots & & \ddots \\ & & & 1 & & & 1 & & 1 \end{bmatrix}$$

Use even allocation matrix A to multiply m×1-dimension Task matrix and then the Tasks would be distributed to n Workers, i.e.:

$$\begin{bmatrix} 1 & & & 1 & & & 1 & & \\ & 1 & & & 1 & & & 1 & \\ & & \ddots & & & \ddots & \cdots & & \ddots \\ & & & 1 & & & 1 & & 1 \end{bmatrix} \begin{bmatrix} \text{task}_1 \\ \text{task}_2 \\ \cdots \\ \text{task}_m \end{bmatrix} =$$

$$\begin{bmatrix} \text{task}_1 + \text{task}_{n+1} + \cdots \\ \text{task}_2 + \text{task}_{n+2} + \cdots \\ \cdots \\ \text{task}_n + \text{task}_{2n} + \cdots \end{bmatrix}$$

As specified above, the allocation results would be obtained. Task_1,Task_{n+1}, and so on, is allocated toWorker_1. Task_2,Task_{n+2}, and so on, is allocated to Worker_2.Following the models and algorithms specified above, tasks would be allocated to Workers for execution in even distribution method. In addition to the basic allocation matrix provided by the platform, users could also set up allocation matrices and allocate algorithms based on their own need, enabling the users to flexibly realize task scheduling and allocation according to their demand.

The scheduling models and allocation methods could also be expanded, such as attributing weight to the a_{ij} in Task allocation matrix A; another example is, the number of connection of Workers could be used for even distribution to make network traffic more even and more efficient. In this case, the adjacent matrix of the Task could be denoted as T, then A is a m×m-dimension matrix:

$$\begin{bmatrix} t_{00} & \cdots & t_{0m} \\ \vdots & \ddots & \vdots \\ t_{m0} & \cdots & t_{mm} \end{bmatrix} \quad t_{ij} = \begin{cases} 1 & \text{Task[i] is Task[j]'s upstream} \\ 0 & \text{else} \end{cases}$$

Then the number of connections of No. k Worker is:

$$\sum_{y=0}^{m} T(x,y) + \sum_{x=0}^{n} T(x,y) \qquad a(x,k) = 1 \text{or } a(k,y) = 1$$

Users could choose proper allocation algorithms according to the type of applications to make the computing, memory occupation and traffic load as even as possible, increasing the usage rate of the platform's resources.

D. Dynamic Load Balancing

Balanced assigned tasks are able to run due to task scheduling model, in the Task level, this can guarantee the balance of the task allocation. However, the balanced distribution of traffic has a huge impact on the performance of the platform too. For example, in general, the traffic of a worker with 2N tasks is twice as much as the traffic of a worker with N tasks. Besides, if some nodes with the worker runs other tasks at the same time, we cannot control the traffics flow through these nodes. Load balancing is hard to reach.

The difficulty of load balancing about many-many flow in distributed streaming platform is:

1) Distributed session. A number of Tasks in upstream Bolt may have the same session data, these distributed data need to be globally alignment. So we need to have a global load balancing scheduler.

2) Cannot use global session table. Because of the large number of nodes, the traffic is very large, if each node inquiry the global session table before sending data, the global session table is easy to overload and become the bottleneck of the platform.

3) Session consistency. In the absence of a global session table, the consistency of the session is very difficult.

Paper[21] based on distributed hash table (DHT) of hierarchical network structure design to solve load balancing problems. On the basis of that, SpeedStream design a framework of dynamic load balancing about many-many flow to solve the problems.

The mechanism of our traffic load balancing is different from the traditional traffic load balancer. Traditional load balancer aggregate traffic on the device and then distribute. Our traffic load balancing is a scheduler which can be dynamically deployed together with the cloud task, and can be seen as a special kind of cloud task. The main idea is to prepose the service table to the upstream task, the connection table is set to the downstream task. The traffic does not pass through load balancing scheduler. Traffic are distributed by the service table on the upstream task. Load balancing scheduler is only responsible for migration and synchronization of service entries. In the synchronization period, relying on the forwarded between the downstream tasks to ensure consistency session. In particular in the following points:

1) Using scheduler with state, the scheduler will interact only if the back-end downstream node's state has been changed, so the scheduler with state will not be the bottleneck. Scheduler failure does not affect the business operation, thereby improving the availability of load balancing system.

2) Using distributed session table, that means every downstream back-end node store its own session table. Scheduler query back-end nodes to check out the session's state(active or passive) to decide whether the session can be migrated.

3) The back-end nodes forwarding data to each other due to the session table to ensure the session consistency of upstream node's during the synchronization phase.

An example is shown in Fig. 4:

1) If we found that the Task4 is overloaded, we should find a downstream Task which is not overloaded to share the load.

2) Assume that Task5 is the destination node to share the load of Task4. Plan to migrate Mod100=41:50 data items to Task5.

3) Query the connection table on Task4, there is an active connection(Mod47), so migrate inactive data segment of 48:50 data items to Task5.

4) Establish a forwarding table on Task4 to transmit the inconsistent data in synchronization period of the service table on upstream tasks.

5) To inform all service table on upstream tasks that the data segment 48:50 has been migrated and the table service table need to update. In the process of update, all the upstream tasks will have a period of inconsistency. During this period, forwarding table on Task4 forward the data which comes from upstream tasks which has old service table to Task5. This can ensure data consistency during period of load balancing.

6) After all service table on upstream tasks complete synchronization, delete the forwarding table on Task4.

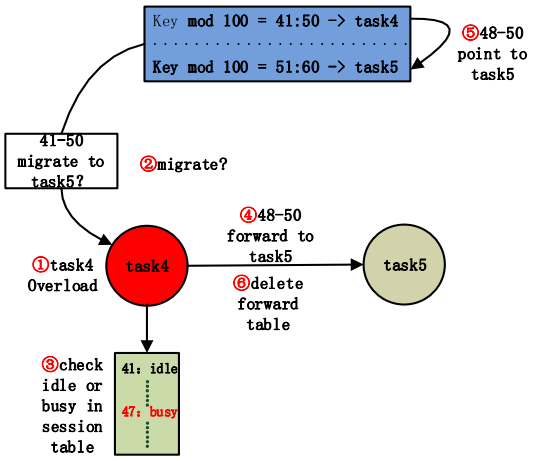


Fig. 4. Steps of dynamic load balancing

E. High-available Design

In this paper, we designed a high available framework. As shown in Fig. 5, all nodes in the cluster have no state, which is able to detect failure of process level and node level timely and accurately. We also apply failure recovery tactic to ensure security of running business in our platform.

The main superiority of the framework is as follows:

1) Stateless framework

All states information of whole clusters are stored in reliable Zookeeper system. There are no control message between nodes. Nodes' failure can't cause the loss of states and can't affect other nodes. We don't need IP spoofing when recovering fault because of stateless nodes.

2) Fault-tolerant for process level

We can ensure security of real-time business by Supervisor→Worker→Task monitoring mechanism. Supervisor can detect and restart Worker timely when Worker break down. Worker can also detect task's fault timely, restarting and recovering tasks.

3) Fault-tolerant for node level

The fault of Supervisor and physical nodes can be reported to Zookeeper by heartbeat. Master will discover the failure nodes and tasks of failure nodes can be migrated to others timely, by monitoring the heartbeat of every node in Zookeeper. Masters itself adopt the way of competing distributed lock, the one which gets the lock can work. Which ensures the high availability of master node.

4) Any node can breakdown

It can't affect platform's normal work, if any node breakdown in our platform. Fault-tolerant rate of every module: a. Zookeeper node: maximum number of nodes fault is 50% of total nodes. b. Supervisor node: maximum number of nodes fault equal to the number of spare nodes. c. Master node: as long as there is one master node alive.

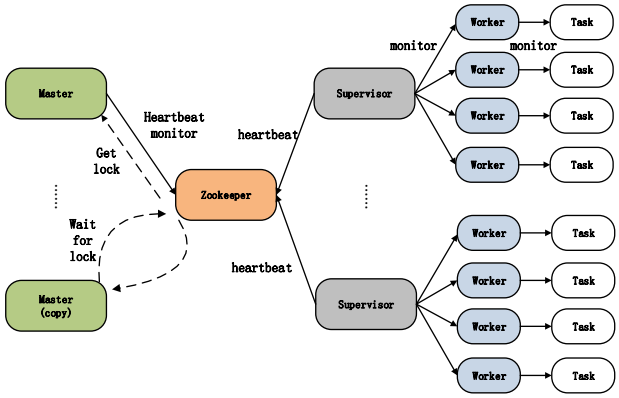


Fig. 5. High-availabel Design

V. EXPERIMENT

Experiment's environment and deployment of cluster is shown in Fig. 6. The cluster contains eleven servers, three of them are cluster of Zookeeper, five of them are job nodes of Supervisor, two of them are backups of Master nodes, and one is client node.

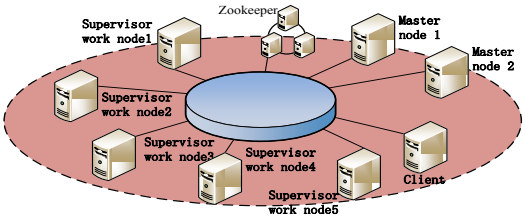
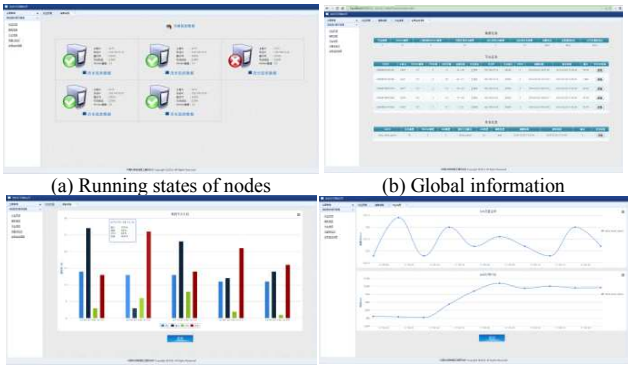


Fig. 6. Cluster deployment

From SpeedStream's web monitor interface we obtain data to draw figures and curves in the following sections. We have designed and implemented a set of interfaces to monitor the platform's running state. Fig. 7(a) shows us the interface of running states of computing nodes in the cluster. For example, from Fig. 7(a) we can see one node is down and another is idle. Fig. 7(b) tells global information including IP addresses of nodes and ports of workers, etc. Fig. 7(c) tells each node's current workload and Fig. 7(d) shows throughput of nodes.



(a) Running states of nodes (b) Global information

(c) Workload of nodes (d) Throughput of nodes

Fig. 7. Graphic user interface of SpeedStream

A. Resource Utilization

Table I and Table II indicates that, under the conditions of no business running on the platform, Nimbus process of Storm platform occupy 110MB of memory resources and 0.7% CPU usage, Our Master process occupy only 1.6MB of memory resources and 0.4% CPU usage; Supervisor process of Storm platform occupy 194MB of memory resources and 0.2% CPU usage, Our Supervisor process occupy only 8.9MB of memory resources and 0.1% CPU usage. So, SpeedStream has better performance than Storm in terms of resource utilization.

TABLE I. RESOURCE UTILIZATION OF STORM

	Nimbus	Super-visor0	Super-visor1	Super-visor2	Super-visor3	Super-visor4
CPU(%)	0.7	0.2	0.2	0.2	0.2	0.2
MEM(KB)	120620	224020	182824	157228	215780	217924

TABLE II. RESOURCE UTILIZATION OF SPEEDSTREAM

	Master	Super-visor0	Super-visor1	Super-visor2	Super-visor3	Super-visor4
CPU(%)	0.4	<0.1	<0.1	<0.1	<0.1	<0.1
MEM(KB)	1716	9524	8244	10035	9033	8564

B. Data Processing Delay

This experiment will compare the data processing delay between Storm and SpeedStream with the same throughput. First, adjust the parallelism and packet sent interval of the Spout(data source) to make the two platform produce 30MB data per second; then, compare the data processing delay. In Fig.8, we can see that the data processing delay of Storm lies from 40ms to 50ms, and the data processing delay of SpeedStream lies from 20ms to 30ms. So, SpeedStream has better performance than Storm in terms of data processing delay.

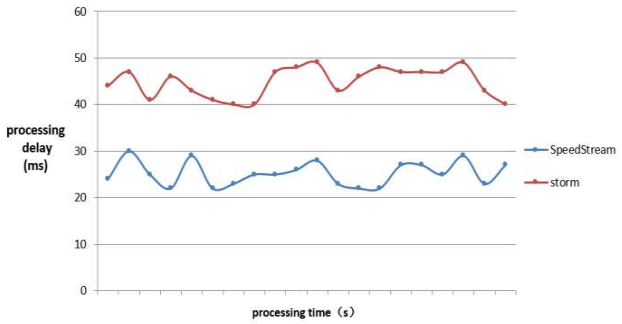


Fig. 8. Data processing delay comparison

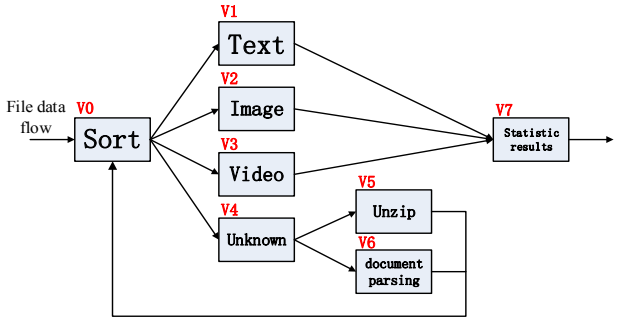


Fig. 9. Job Example

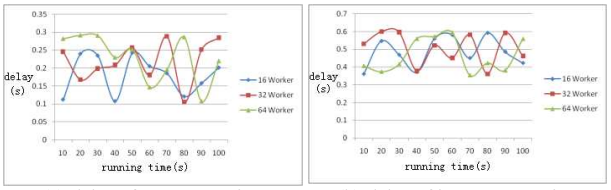
C. Computing model

The existing stream computing model is lack of support for iterative and feedback stream computing business. This experiment processes real data flow in a production environment: the processing of different type of data stream. This will demonstrate the advantage of the computing model's support for iterative and feedback stream computing business.

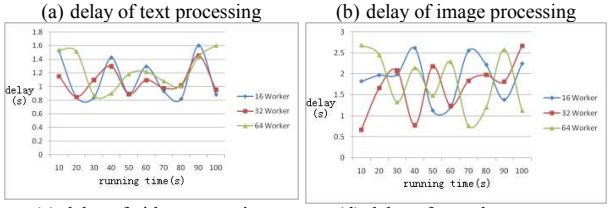
As shown in Fig. 9, there are 8 vertices in the directed graph with sources and sinks. When data flowing through V0, according to the type of document, the file data is distributed to V1, V2 and V3. If the type of the file data is unable to be directly obtained, the file data will be passed to V4. V4 will check the file and pass it to V5 and V6 which will analyze files and return back the file type to V0 again. The loop iteration limit of V0-V4-V5V6 is set to 3, which supports a maximum of three layer file type resolving. The files which have flown through V5 and V6, but still unable to be processed, will be discarded.

Fig. 10 shows the delay with different amount of Workers, representing text processing (Fig.10(a)), image processing

(Fig.10(b)), video processing (Fig.10(c)) and unknown format document processing (Fig.10(d)). The processing delay increases with the file size and complexity. The average delay of text processing is within 0.3 seconds, the average delay of image processing is within 0.6 seconds, the average delay of video processing is no more than 1.6 seconds, due to the presence of the iterative process, the processing delay of an unknown format document is volatile, which is 0.6-2.7 seconds.



(a) delay of text processing (b) delay of image processing

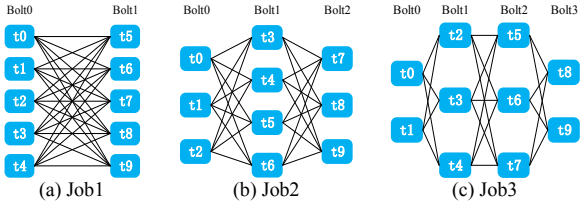


(c) delay of video processing (d) delay of an unknown type document processing

Fig. 10. Processing Delay

D. Scheduling and Load Balancing

We compared our approach and algorithms built-in Storm and S4 and valuating indicators are load balance of nodes and communication traffic through switches.



(a) Job1 (b) Job2 (c) Job3

Fig. 11. Topology structures of jobs

We submit a job selected randomly from the three jobs every 20 seconds. These jobs' topology structures are illustrated in Fig.11. Then we record the workload of nodes and communication traffic through switches every 5 seconds.

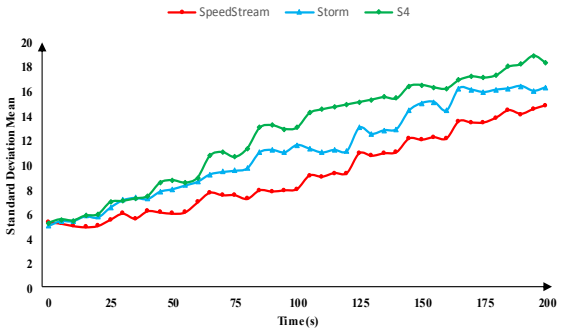


Fig. 12. Workload standard deviation mean

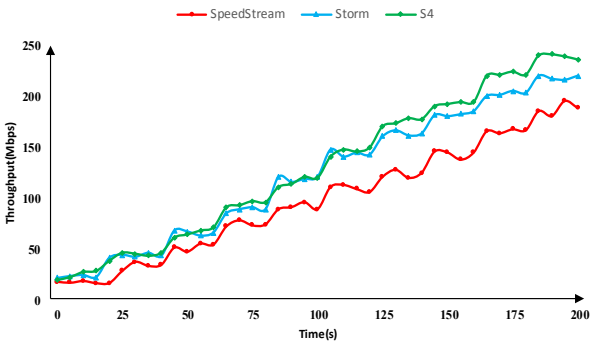


Fig. 13. Communication traffic through switches

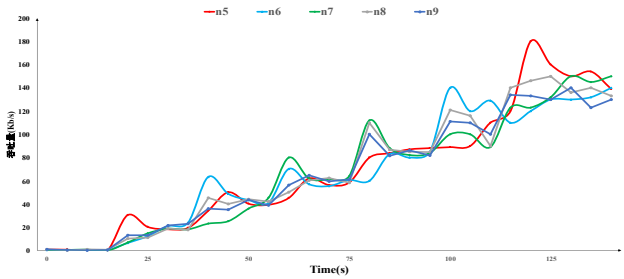


Fig. 14. Data traffic distribution

The vertical axis in Fig.12 means workload standard deviation mean of nodes, formulated mathematically as follows.

$$S = \frac{1}{2n}\sum_{i=1}^{n}\sqrt{(cpu_i - \mu_{cpu})^2 + (mem_i - \mu_{mem})^2}$$

$$\mu_{cpu} = \frac{1}{n}\sum_{i=1}^{n} cpu_i$$

$$\mu_{mem} = \frac{1}{n}\sum_{i=1}^{n} mem_i$$

n is the number of nodes in the cluster and n=5 in this experiment. The variables μ_cpu and μ_mem are mean of load of CPU and memory of nodes in the cluster. S depicts the degree of load balance of all nodes.

Curves in Fig.12 show how the standard deviation mean varies with time. As the number of jobs running in these platforms continues to grow, standard deviation mean tends to increase. With the same jobs running, standard deviation mean of SpeedStream is less than that of Storm and S4. That is, load balance of SpeedStream is better.

Curves in Fig.13 show how communication traffic varies with time. The more jobs we submit to the cluster, the more communication traffic flows through switches for all three platforms. Most of the time, the curve representing SpeedStream is below the other two, which means less stream data flows through switches when scheduling by our presented approach. This is because enough consideration for the relationship of nodes is given when our approach performs during scheduling progress. That is, relevant tasks are assigned to the same node or

adjacent nodes and traffic through switches reduces, thus relieving the bandwidth pressure of the cluster.

Fig.14 shows the data traffic distribution of Bolt1 in Job1. Bolt0 increases data traffic every 20 seconds. With the increase of data traffic in Bolt0, the traffic in Bolt1 also increases. First, the traffic in Bolt1 is not balanced after the traffic increase. Then, the traffic in Bolt1 becomes balanced due to the load balancing strategy. This is because the scheduler will transfer the data traffic due to the flow fluctuation to achieve load balancing under the condition of maintaining consistency of session, which reduces the pressure of peak flow traffic.

VI. CONCLUSION

SpeedStream is a universal distributed platform that can handle with massive data flows with the features of low coupling, high availability, low latency and high scalability. The main contribution of SpeedStream are as follows: 1) Aiming at the availability of real-time streaming computing platform, we design a high availability framework basing on Zookeeper. It ensures timely detect and repair fault of process level and node level by monitoring heartbreak information of each modules and applying the strategy of fault migration. 2) In order to reduce the excessive requirements for business model of traditional platform, we design a real-time streaming computing model that depends on directed graph with sources and sinks, which can not only satisfy the needs of common DAG computing service, but also support iteration computing services including directed cycle, bidirectional arcs and annular arc, and feedback flow computing service by directed cycle detection and iteration protection. 3) The platform can realize personalized task scheduling strategy for users by establishing task allocation matrix and optimize task allocation model. 4) In order to solve the many-to-many dynamic load-balancing between tasks, we adopt scheduler of status level and technology of distributed session table. It overcomes the difficulty of maintaining consistency of session without global session table, and testify that the convergence of this method that can balance resources and improve the utilization of resources. This platform provides reliable, universal, and quick solutions to process massive data flows in real time, such as to process the real-time trading data in e-commerce, to analyze sensing flow in internet of things, and monitor flow of the Internet.

REFERENCES

[1] Yogita Y. and Toshniwal D. Clustering techniques for streaming data-a survey. Advance Computing Conference (IACC), 2013 IEEE 3rd International, Feb.2013:951-956.

[2] Lakshmi K.P and Reddy C.R.K. A Survey on different trends in Data Streams. Networking and Information Technology (ICNIT), 2010 International Conference, June 2010:451-455.

[3] Pitman A. and Zanker M. Insights From Applying Sequential Pattern Mining To E-Commerce Click Stream Data. Data Mining Workshops (ICDMW), 2010 IEEE International Conference, Dec.2010:967-975.

[4] Martinez-Julia P., Torroglosa Garcia E., Ortiz Murillo J., et al. Evaluating Video Streaming in Network Architectures for the Internet of Things. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference, July 2013:411-415.

[5] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[6] Dean J, Ghemawat S. MapReduce: a flexible data processing tool[J]. Communications of the ACM, 2010, 53(1): 72-77.

[7] Abadi D, Carney D, Cetintemel U, et al. Aurora: a data stream management system[C]//Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM, 2003: 666-666.

[8] Abadi D J, Ahmad Y, Balazinska M, et al. The Design of the Borealis Stream Processing Engine[C]//CIDR. 2005, 5: 277-289.

[9] Condie T, Conway N, Alvaro P, et al. MapReduce Online[C]//NSDI. 2010, 10(4): 20.

[10] Twitter Storm: http://storm-project.net/.

[11] Zhengping Qian, Yong He, Chunzhi Su, et al. TimeStream: Reliable Stream Computation in the Cloud. EuroSys, 2013:1-14.

[12] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[C]//Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 2010, 10: 10.

[13] Neumeyer L., Robbins B., Nair A., et al. S4: Distributed Stream Computing Platform. Data Mining Workshops (ICDMW), 2010 IEEE International Conference, Dec.2010:170-177.

[14] Zaharia M, Das T, Li H, et al. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters[C]//Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing. USENIX Association, 2012: 10-10.

[15] Zookeeper: http://zookeeper.apache.org/.

[16] Hunt, Patrick, et al. ZooKeeper: Wait-free Coordination for Internet-scale Systems. Proceedings of the 2010 USENIX conference on USENIX annual technical conference. Vol. 8. 2010.

[17] Zhao L, Chuang Z, Ke-Fu X, et al. A Computing Model for Real-Time Stream Processing[C]//Cloud Computing and Big Data (CCBD), 2014 International Conference on. IEEE, 2014: 134-137.

[18] Schmidt S., Legler T., Schaller D., et al. Real-time Scheduling for Data Stream Management Systems. Real-Time Systems, 2005. (ECRTS 2005). Proceedings. 17th Euromicro Conference, July 2005:167-176.

[19] Leidi T., Heeb T., Colla M., et al. Event-driven Scheduling for Parallel Stream Processing. Parallel Computing in Electrical Engineering (PARELEC), 2011 6th International Symposium, April 2011:36-41.

[20] Meng-Meng C, Chuang Z, Zhao L, et al. A Task Scheduling Approach for Real-Time Stream Processing[C]//Cloud Computing and Big Data (CCBD), 2014 International Conference on. IEEE, 2014: 160-167.

[21] Lin Ouyang, Qing-ping Guo. A Dynamic Load Balancing Technique of Distributed Stream Processing System. Future Generation Communication and Networking Symposia, 2008. FGCNS '08. Second International Conference, Dec.2008:52-57.