The Journal of Supercomputing (2023) 79:21028–21056
https://doi.org/10.1007/s11227-023-05473-z

A two-tier coordinated load balancing strategy over skewed data streams

Dawei Sun[1] · Minghui Wu[1] · Zhihong Yang[1] · Atul Sajjanhar[2] · Rajkumar Buyya[3]

Accepted: 1 June 2023 / Published online: 18 June 2023

Abstract

Load imbalance severely affects cluster performance, and the polarization of resources due to load skewing leads to further worsening of system throughput and latency problems. The proliferation of tasks to be processed in the big data era leads to more severe load skewing. How to cope with the surge of skewed data stream in the context of big data is a new challenge now. In this paper, we propose a coordinated load balancing strategy on skewed data streams (referred to as St-Stream), which is a two-tier hierarchical system for handling data streams. The proposed strategy is characterized by performing a migration pairing strategy for resources at the task allocation stage by cutting and moving out the tasks of high-load nodes in a hierarchical manner, and the moved-out operators are placed in the routing table, and the routing table operators are moved out to these nodes sequentially according to the tasks required by low-load nodes. We further design a two-tier coordination scheme for the resource allocation problem, which can adjust the skewed load from within the nodes and then dynamically restore the balance between the nodes. We implemented St-Stream on Apache Storm, which achieves a 21% coordination in processing CPU utilization, a 17.6% reduction in latency, and a 0.3 improvement in load balance recovery compared to the baseline design. Our experimental results demonstrate that the proposed load balancing strategy better balances the cluster load and improves the performance of the stream processing system.

Keywords Apache storm · Distributed systems · Load balancing · Skewed data stream · Stream computing · Two-tier coordination

✉ Dawei Sun
sundaweicn@cugb.edu.cn

[1] School of Information Engineering, China University of Geosciences, Beijing 100083, People's Republic of China

[2] School of Information Technology, Deakin University, Geelong, VIC 3216, Australia

[3] The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

1 Introduction

Data streams processing has become an essential component of Big Data, which mainly provides a millisecond response for processing continuous and unbounded data. However, data with the same characteristics are aggregated to the same task for processing, which may lead to uneven distribution of data between tasks and further cause the so called data stream skewing phenomenon. Skewed data stream is one of the main factors causing the load imbalance problem and affecting the performance of distributed processing systems [1–3]. In addition, the unpredictable size and speed of the data received by the stream processing system leads to its processing in a very unstable manner [4, 5] causing load fluctuations in the system. Thus, the skewed data streams can incur two problems: (1) From the perspective of external data stream into the processing system, the arrival of large-scale uncontrollable skewed data streams has different capacities within the cluster to withstand the load [6, 7], leading to the paralysis or even collapse of the whole system. (2) From the perspective of the system, data are randomly assigned to nodes [8], and the communication overhead between nodes varies due to the different speed and capacity of each node to handle the task [9], making the skewed data streams enter the system at different scales leading to system crashes.

At this stage, the impact of skew load on data stream on latency, CPU utilization and node overhead is not sufficient. Combining big data technology with load balancing technology can further provide specialized services that guarantee network security, improve network efficiency, and reduce network latency [10]. [11] present two path selection solutions based on the overall load of the network to find the optimal paths considering the network state. When dealing with large-scale data, according to its characteristics and the rich computing power of the stream computing strategy [12], it is not enough to consider only a single-tier scheduling of resources. Instance scheduling within a node is related to data stream scheduling, and data streams are associated with other relevant nodes. Therefore, considering only the instance scheduling within a node may produce isolated resources due to the ignorance of other nodes' resource load. Moreover, considering only inter-node scheduling without instance scheduling cannot sufficiently exploit the performance of the system. All these factors inspire us to propose a two-tier scheduling strategy as an effective solution to the current problems. In a distributed environment, the utilization of resources such as memory, CPU, and disk by nodes in a cluster is the main basis for load balancing of the system. Each node in the system has different capacity and speed to handle tasks [13], and the allocation to them is random, which can lead to some nodes running under high load for a long time and another part of the nodes' resources being idle for a long time, generating a load skewing phenomenon, as shown in Fig. 1 The density curve of skewed load is concave and convex, and the more convex at the highest point represents a more serious load skewing situation of the cluster, while a balanced load density curve presents a smooth and dispersed state, so the load skewing situation is expressed by the load density.

Stream computing systems have been an important part of research and are applied to multi-domain processing problems. To ensure the data processing

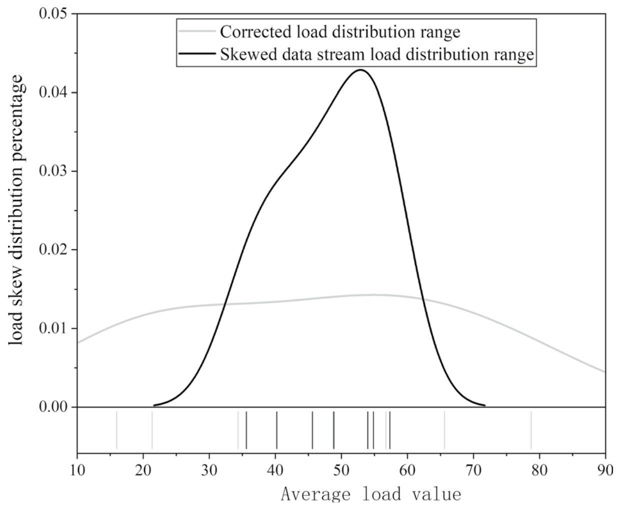


Fig. 1 Skewed density curve

capability of the online distance education platform, the cloud computing architecture is used to solve the bottleneck problem of data processing [14] and improve the stability of data and user experience. In [15], a stream computing-based approach is proposed for in-situ sensors and remote sensing observations, which can provide timely decisions for natural disaster management and implementation computation for heap flooding information.

Load balancing strategies have been extensively studied in distributed systems. In [16], the authors explore the relationship between linear time series and load and find that load has a high correlation over time, which suggests that load can be predicted using time as a scalar. In [17], while satisfying the persistence, dynamics and unpredictability of data streams, the authors find that the communication overhead from load migration can be minimized by selectively activating data at high-load nodes and replicating them at low-load nodes; Another Hot-Key-based approach is [18], which proposes a novel load balancing mechanism, fish, where the authors find that the time-varying Key of a data stream can form a skewed distribution within a fixed time interval.

The network data transfer rate is also closely related to the workload, and the proposal in [19] fixes the routing path at the beginning of the transfer phase and migrates many data streams on the network to the unused heavy resource network by balancing detection, but it only considers a single-layer migration strategy, which is not sufficient when resource allocation problems occur. Because the requests produced by IoT devices may vary over time and we need to find a dynamical policy one which can tell how to allocate resources in different time periods [20].

To summary in the proposed St-Stream strategy, we focus on dynamic pairwise migration of skewed data streams, considering a two-tier coordination strategy within and between nodes to handle the skewed load.

1.1 Motivation

In a distributed environment, the resource utilization of nodes in a cluster such as memory, CPU, disk, etc. is the main basis of whether the system load is balanced or not, while the uneven deployment of massive data in a cluster will lead to some nodes running under high load for a long time and another part of node resources being idle for a long time. In addition, our experiments show that the system has a relatively good performance under balanced load on the system. However, the system has a longer response time and lower throughput, when there is a skewed data stream causing an unbalanced load.

Some scheduling algorithms [4, 21] attempt to balance the resource load in the cluster by scheduling the deployed tasks. However, balancing the resource load by scheduling tasks brings an additional costly overhead to the system. To address this problem, recent works [22–24] propose a data stream scheduling to reduce the scheduler's overhead. These efforts attempt to balance the resource load between deployed tasks, but they ignore the fact that the nodes running multiple tasks have different processing capacities, resulting in unbalanced resource load in the cluster.

In a skewed data stream environment, it is not enough to only consider the balanced resource load between nodes, because unbalanced load on different tasks of the same node can increase the system response time. The compute node acquired by tasks is a multi-core environment. If there is load imbalance between tasks deployed on the same compute node, the tasks with high load may suffer data loss and have longer queuing time, which in turn affects the performance of system. Moreover, when severe load imbalance is sustained over a long period of time, the tasks with high load may experience downtime. Therefore, the load imbalance between tasks on the same compute node can affect the performance of system. It is also not enough to only consider the balanced load between tasks of one node, because unbalanced load between nodes can also affect the performance of the system. So, a two-tier coordinated load balancing Strategy is needed by considering both the intra-node and inter-node load balancing so that the full potential of the system computing resource can be reached.

1.2 Contributions

In view of the state-of-the-art solutions and associated problems, we propose a two-tier coordination load balancing strategy over skewed data streams to solve the load balancing problem in distributed stream computing systems due to skewed data streams. In this paper, we make the following contributions.

(1) We have conducted a comprehensive investigation of the factors that affect the system load, based on the skewed data, and proposed a load estimation criterion based on CPU utilization.
(2) The St-Stream optimization algorithm is proposed to use a migration strategy for intra-node and inter-node arithmetic to balance the load between nodes.

Table 1 Related work comparison

	Related work						
Parameter	[20]	[23]	[24]	[25]	[26]	[27]	St-stream
Prediction modeling	✗	✓	✓	✓	✗	✗	✓
Performance modeling	✓	✓	✓	✓	✓	✓	✓
Multi-level coordination strategy	✗	✗	✗	✗	✗	✗	✓
Reource saving	✓	✓	✓	✓	✓	✓	✓

(3) We evaluate the throughput and latency of St-Stream's runtime policy. The experimental results show that coordination in processing increases the CPU utilization by 21%, reduces the latency by 17.6% and improves the load balance recovery by 0.3 compared to the baseline design.

The rest of the paper is organized as follows. Section 3 defines the communication model, resource load model and load offset repair model. The implementation method and algorithm of St-Stream are proposed in Sect. 4. Section 5 describes the experimental environment, parameter settings and performance evaluation of St-Stream. Finally, conclusions are presented.

2 Related work

In this section, we review recent work in two related areas: load balancing of nodes and data stream partitioning strategies. The comparison between our work and other closely related work is summarized in Table 1.

2.1 Load balancing

Load balancing between nodes in distributed systems plays an important role for improving the performance of system. Finding an optimal schedule for load balancing between nodes is proved to be NP-hard [28]. Because of its theoretical and practical importance, it has been studied extensively over the years and continues to be a topic of research.

Cluster load imbalance may be caused by the scheduler ignoring the allocation dependency between time slots. In [26], the authors assigned time slots uniformly to multiple topologies in the cluster and used a merge factor to eliminate failures and restore the cluster balance. If this approach is used to handle the data after isolated scheduling, load skewing still occurs and this problem has not been adequately addressed.

To implement load balancing in distributed stream computing environments, a TTNS algorithm [29] was proposed to increase the utilization of cluster resources. It mainly balances the load between nodes by classifying task types and dynamically monitoring node resources.

To distribute workload upon nodes evenly, a resource-oriented load balancing task scheduling (RoLBTS) mechanism [30] for distributed stream computing systems was proposed. Based on the barrel principle, it respectively selects the memory and the bandwidth to model the resource occupancy ratio of the physical node and that of the physical link. Based on the self-recursive calling, the RoLBTS algorithm is applied for task resource scheduling.

To solve the problem that the load of nodes in stream computing system fluctuates drastically, a load prediction-based elastic resource scheduling strategy [31] was proposed. It first identifies the performance bottleneck and resource redundancy of the cluster and proposes a resource scheduling algorithm to draw up the rescheduling plan. An online load migration algorithm is then executed to adjust and migrate the processing load of nodes.

In conclusion, if only considering the load balance between nodes, it is not enough to improve the performance of system. The above solutions provide valuable insights into the challenges of load balancing in a distributed stream environment. However, in the era of big data, new approaches must be developed to deal with the unique challenges, and some characteristics specific to distributed stream system need to be considered.

2.2 Data partitioning

Data partitioning means that data tuples are scheduled to tasks in random or hash ways. A poor data partitioning strategy has a negative impact on the performance of system. Therefore, load balancing of data partitioning between parallel instances in distributed stream computing systems has been widely studies in recent years.

To deal with the uneven load caused by skewed data, PKG [27] used both key-value partitioning and local load evaluation solutions to adapt to the skewed data streams. It first maps the tuple using two Hash functions to select two candidate downstream tasks. Next, the tuple is emitted to a less loaded task using local load evaluation. When the load of both candidate tasks is high, it is not efficient to solve the unbalanced load between tasks because the load of the other tasks may be low. In [32], the authors proposed to allocate hot keys and automatically adjust hot keys without routing tables to maximize the memory and computational cost of replication and ground to ensure load balancing.

To implement the load balancing of inter-task, difference-aware load of task in distributed stream processing systems [33] was proposed. It uses the random partitioning to assign the tuples with high frequency and the hash function to assign the tuples with low frequency. The tuples with high frequency can be identified by weighted probability counts and the estimated tuple probability can change with time.

In [34], a sketch-based pre-filtered partitioning algorithm (PFG) was proposed. On one hand, to ensure better load balancing for the skewed data streams, the

Table 2 Description of main symbols

Symbol	Description
S_i	The i^{th} compute node
$M(f_{ik})$	The k^{th} CPU frequency in node S_i
$M(m_i)$	Memory size of node S_i
$L(S_i)$	Resource load of node S_i
$W(S_i)$	Load weight of node S_i
α_i	Offset degree of node
θ	Load range constraint threshold

detected hot keys are directed to more than two candidate tasks randomly. On the other hand, for less frequent keys, the proposed solution explores the principle of the power of two choices to distribute load. For the less frequent tuples, they are assigned directly to the two candidate instances. PFG uses local load estimation to select the target task with the least data to be processed. PFG uses local load estimation to select the target task, whose load is the lowest.

However, when the tuples with low frequency are aggregated to the same compute node, load imbalance between nodes can be caused. Therefore, it is necessary to implement a two-tier load balancing strategy to balance the load between and within nodes.

3 System model

This section focuses on the communication modes, resource load model and load offset repair model. For the sake of clarity, in Table 2, we summarize the main notations used throughout the paper.

3.1 Communication model

A streaming application can be modeled as a directed acyclic graph (DAG). A task emits its processed data to the direct successor tasks by random or key-value grouping. There are communications between the task and the successors. Moreover, the communication between tasks is different because of skewed data streams. In stream computing systems, large-scale data of all types often enter the system very erratically at an unpredictable scale and speed, and this data may lead to load skewing of the entire system [35]. The efficiency of load balancing algorithms depends heavily on the nature of the tasks handled by the system. The communication overhead between nodes and the resource consumption within the nodes have a significant impact on the performance of Storm [36], the higher the load on the nodes, the longer the average processing time of the graph elements and the corresponding decrease in the performance of the system.

The distributed system randomly allocates resources to each node, each node processes data at different rates, and individual nodes generate unbalanced loads, which

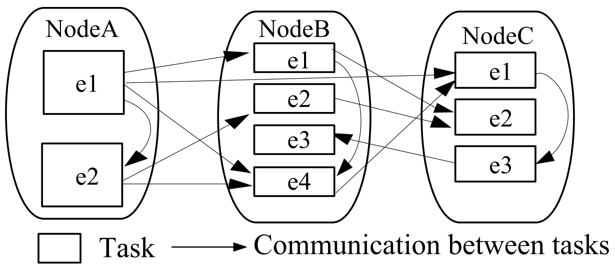


Fig. 2 Intra-node and Inter-node communication

may lead to system crashes. As shown in Fig. 2, there are two main cases that cause the unbalanced load on the nodes. (1) If the task distributes data by Key value, the distribution of data will be uneven due to the differences in the data properties. It further leads to generating unbalanced load in intra-nodes and inter-nodes. (2) If the tasks distribute the data evenly, unbalanced loads in intra-nodes and inter-nodes may also be generated because of the fact that the computational resources consumed by each piece of data are different. In addition, the computational resources consumed by each piece of data are difficult to predict and control. If the system is not intervened at runtime, the load skewing occurrence in the cluster is unavoidable.

3.2 Resource load model

The existing load metrics handle data in small scale with fixed algorithms [37], and the research on load balancing focuses on how to handle the fixed load of nodes in the cluster with high subsequent maintenance cost. Considering the current load of each node in the cluster, the system makes load decisions for each node based on the feedback information from the nodes in the cluster [38]. The data processing time of nodes can be affected by their CPU utilization and memory usage [39]. Therefore, the resource consumption of nodes is measured based on these metrics. In addition, these nodes with high resource consumption can reduce the throughput and increase the response time of the system [16]. With limited computing resources, making the cluster handle huge amounts of data is a challenge. Previous studies [17, 18, 40] also demonstrated that reducing the difference in CPU utilization and memory consumption among nodes could effectively address this problem. Therefore, the analysis of these performance metrics is necessary for solving the uneven load problem caused by skewed data stream in the cluster.

Each load within the cluster is monitored using CPU utilization and memory size. Let the size of cluster be n, the number of CPU cores of each node be u, the CPU frequency of node S_i be $M\left(f_{iu}\right)$ and the memory size of node S_i be $M\left(m_i\right)$, then the performance of node S_i is represented by (1), where $i = 1, 2, ..., n$.

$$M\left(S_i\right) = u \cdot M\left(f_{ij}\right) + M\left(m_i\right) \tag{1}$$

Let the CPU usage frequency be $L(f_i)$ and the memory usage frequency be $L(m_i)$, then the load of node S_i is expressed by (2), where $i = 1, 2, \ldots, n$.

$$L(S_i) = L(f_i) + L(m_i) \tag{2}$$

From (1) and (2), the load weight $W(S_i)$ of a node can be defined as the ratio of node load to node performance, expressed by (3).

$$W(S_i) = \frac{L(S_i)}{M(S_i)} \tag{3}$$

The difference of the node load is the product of the difference between the load weight of the node before optimization and the load weight after optimization and the performance of the node. The larger the difference of the node load, the better the optimized load balancing method is indicated, expressed in (4).

$$\Delta L(S_i) = \left(W_{before} - W(S_i)\right) \cdot M(S_i) \tag{4}$$

When massive data are processed and stored on a large-scale, the goal is often to have lower cost and higher efficiency, so the resource allocation and data scheduling problems are significant. Rational distribution of data streams is to distribute the large-scale data streams to each node in the cluster as evenly as possible, so that the load of each node in the cluster is not very different from the perspective of task allocation, so that the load can be reasonably distributed and the resources in the cluster can be better utilized. Load balancing is to find a reasonable way to solve the data storage problem, resource allocation problem and task scheduling problem in a cluster to achieve efficient and optimal resource utilization. The essence of the resource allocation problem of load balancing is to select the most suitable resource among multiple resources for allocation to meet the current task demand and to achieve the condition of load balancing within the cluster.

3.3 Load offset repair model

Due to the nature of random distribution of node loads, a portion of nodes fluctuate during cluster operation due to the arrival of data streams, and the loads of these nodes rise steeply [35], creating a skewed phenomenon compared to other nodes. The load variation trend of nodes is closely related to time [41], and when large-scale skewed data arrives, the system distributes these data equally according to the order of input, but the actual size of resources occupied by these data within the nodes varies, and in the limit some nodes' resources are idle for a long time [42], which is the skew phenomenon. Load balancing is required to find out the nodes that produce the offset phenomenon and correct the resources that are offset in time to restore the normal range of resource usage.

The way to deal with this problem is to correct the offset load in time and restore the balance of resources within the node so that subsequent resources are allocated normally. The more resources occupied inside the node, the more serious the offset degree is, and the less resources occupied, the more balanced the resource allocation is. Define the degree of balance of resources inside the node by the offset degree α. Define Sx as the ratio of key values with high occupied slot to all key values within the node, which is the current degree of aggregation of key values. Define Sy as the ratio of all data occupying key values with high slot to all data within the node. Then the offset α can be calculated by Equation (5).

$$\alpha = \frac{S_y}{S_x} \tag{5}$$

Where the larger the offset α represents the more offset the current resources are, that is, the load distribution is not balanced.

By adjusting the number of nodes and offset to observe the load of the cluster, then the degree of load balance L of each node is expressed as the product of the number of nodes and the degree of load balance, and the smaller L means that the cluster load distribution is more balanced. It can be described by the Equation (6).

$$L = \sum_{t=1}^{j} \left(L_J + \sum_{i=1}^{n} \left(\alpha_i \cdot L_{j-1} + \left(1 - \alpha_i\right) \cdot \Delta t \right) \right) \tag{6}$$

The change of node load $\Delta L(S_i)$ is the difference between the load weights of node before and after optimization multiplied by the performance of node, which can be described by Equation (7).

$$\Delta L\left(S_i\right) = \left(W_{before} - \frac{q \cdot L\left(f_i\right) + (1-q) \cdot L\left(m_i\right)}{p \cdot u \cdot M\left(f_{ij}\right) + (1-p) \cdot M\left(m_i\right)} \right) \cdot M\left(S_i\right) \tag{7}$$

where q and p denote the weight between CPU frequency and memory, respectively. Moreover, the larger the $\Delta L\left(S_i\right)$, the more even the load and the more stable the performance for the optimized cluster.

4 St-stream: architecture and optimizer

In this section, we focus on the proposed St-Stream architecture and the two-tier coordination strategy.

4.1 St-stream architecture

The St-Stream system is designed to handle skewed data streams using a two-tier independent coordination load balancing strategy for big data scenarios, thereby optimizing the load of the distributed stream computing system from intra-node to inter-node.

The St-Stream system consists of four components, as shown in Fig. 3. The Ganglia collection system and the database are the data collection components, which are responsible for data collection and initialization. The Nimbus

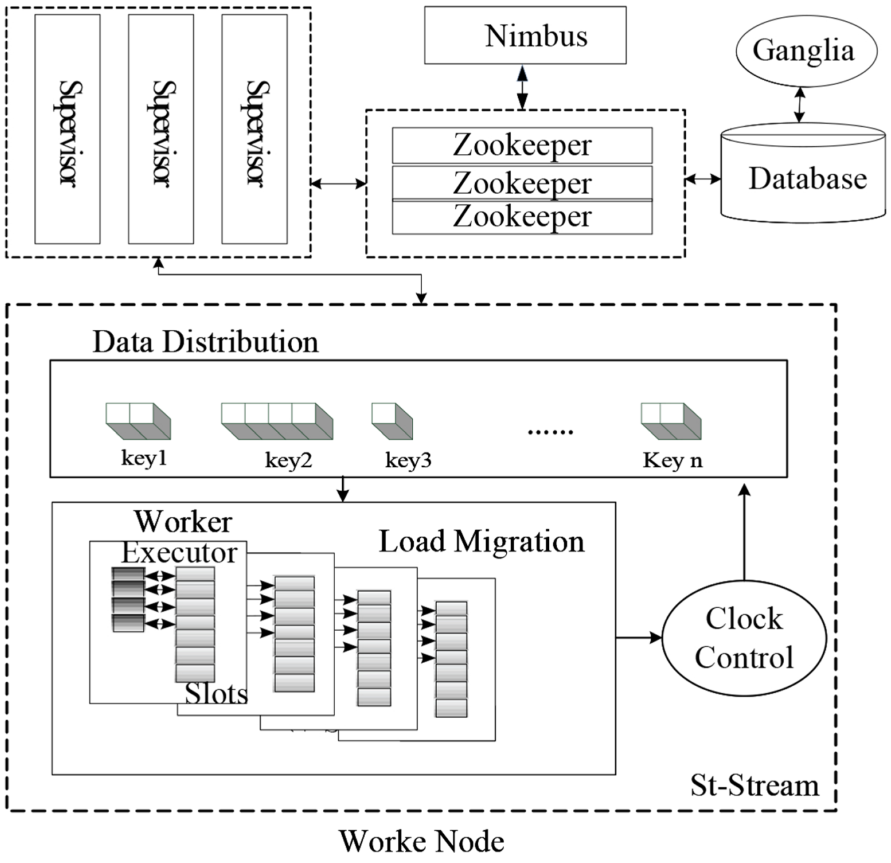


Fig. 3 St-stream architecture

subsystem is used to receive tasks submitted by users and assign these tasks to the supervisors. The Zookeeper subsystem is responsible for performing the communication between Nimbus and Supervisor and monitoring the survival status of nodes in the cluster. The Supervisor subsystem is used to receive the tasks assigned by Nimbus and to manage the worker processes so that Executor traffic exchange takes place in the slots, reducing traffic interaction and communication latency. The worker nodes in the Supervisor are divided into three parts: the data allocation initialization processing unit; the load migration module, which is responsible for dynamically allocating resources and restoring the cluster to normal operation; and the clock monitoring module. The load migration module; and the clock monitoring module, which periodically determines whether migration operations are required to ensure that there is no uneven load while processing tasks.

So, the load problem is handled in two phases in St-Stream. The first phase passes the initialized data from the database and the Ganglia system into the system and restores the overloaded data inside the nodes to the basic range of the load for the difference in the internal processing data capacity of the nodes. [42]. In the second phase the load skewing between individual nodes still occurs, and the skewed load between nodes is optimized to operate without affecting the processing capacity of the system in order to achieve system load balancing and improve the throughput.

4.2 Two-tier coordination of resource allocation

The two-tier coordination strategy can better consider the recovery balance of the skewed load. When the cluster is running, the load variation of the nodes presents a random distribution. Most of the nodes may have little difference in resource load, but there are some nodes that have a steep increase in resource load due to fluctuations in data stream. When starting to input data to the system, the overall system resource load is very low, so the initial load range needs to be given. Since the arrival rate of the data stream is dynamically changing, the normal range of the system load should be dynamically adjustable. The maximum load threshold L_{max} satisfies equation (8) and the minimum Load Threshold Satisfies equation (9)

$$\begin{aligned} &\theta \leq \frac{L_{\max} - L_{avg}}{L_{avg}} \\ &\Rightarrow L_{\max} \leq \theta \cdot L_{avg} + L_{avg} \end{aligned} \tag{8}$$

$$\begin{aligned} &\theta \leq \frac{L_{avg} - L_{\min}}{L_{avg}} \\ &\Rightarrow L_{\min} \geq L_{avg} - \theta \cdot L_{avg} \end{aligned} \tag{9}$$

where θ denotes the load range constraint threshold.

(a) **Intra-node load balancing.** For the resources within the working nodes, the massively skewed data streams are distributed according to the hash distribution strategy, and different slots within the nodes are allocated according to different key values to complete the initial data distribution.

The key to effectively reduce node utilization is to select the operator that occupies the most CPU resources for migration. Set the estimation condition: CPU utilization is in the low load state between $[0,L_{\min}]$, in the general case between $(L_{\min}, L_{\max}]$, and in the high load state between $(L_{\max},1]$.

The initialized data from the upstream input are distributed to the downstream tasks by default using a hash allocation policy, which allocates the same slots within the node for the same key values. If the clock monitoring module detects a skewed cluster load within a clock cycle, St-Stream will migrate and adjust these resources according to the pairing principle to eliminate the uneven distribution of resources generated by the overload. The load skew situation is handled as follows:

(1) Direct move out: Processing units that exceed the high load threshold $L_{\max}$ are directly moved out and placed in the temporary routing table R, as shown in Fig. 4.

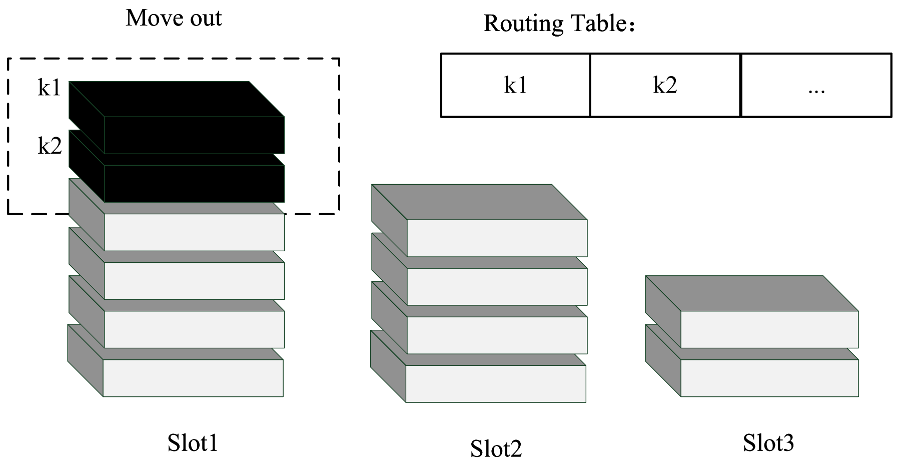


Fig. 4 Direct move out diagram and Move in routing table

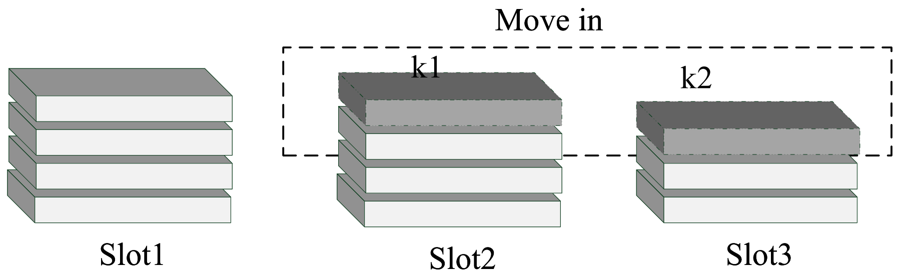


Fig. 5 Move in the key in the routing table

(2) Direct migration: The outgoing processing units stored in the routing table are migrated sequentially into the low-use slots that handle the same tasks, as shown in Fig. 5.

(3) Split migration: If the $L_{\max}$ is still exceeded after the direct migration, the direct migration operation cannot be performed, and the processing unit needs to be split, and the split processing unit will be put into the routing table in order for the indirect migration operation, as shown in Fig. 6.

$$k_{i1} = \forall_{i=1}^{l}\left(k_i - L_{\max}\right) \tag{10}$$

$$k_{i2} = \forall_{i=1}^{l}\left(k_i - k_{i1} - \sum_{i=1}^{l-1} k_i\right) \tag{11}$$

$$S_{(1,2,\ldots l)} = asc\left(\forall_{\mathrm{i}=1}^{l}\left(\mathrm{k}_{i1}, k_{i2}\right)\right) \tag{12}$$

$$Mi_{(1,2,\ldots,l)} = \forall_{i=1}^{l}\left(T - S_i\right) \tag{13}$$

After the move in/out operation of the processing unit within one clock cycle T, all slots inside the node should be below the $L_{\max}$ threshold, and the balance is restored from the load skew state inside the node to resume the distribution of upstream data.

The intra-node load balancing algorithm is described in Algorithm 1. Overloaded processing units are directly paired and moved to the routing table by steps 1–7. The critical processing units are split according to task size and placed in the routing table by steps 8–10. Sort the routing table and migrate into the low-load routing table by steps 11–14. After completing one clock cycle of migration adjustment, the data processing continues for the next moment according to the task state of the current processing unit. The time complexity of algorithm 1 is $O(k \cdot h)$, where k is the number of high-load processing units and h is the number of other processing units.

Algorithm 1: The intra-node load balancing algorithm.

```
   Input: $C = c_i$, Initial data stream;
   Output: Inter-Point tuning strategy.
1  Initialize the routing table $R$;
2  /* Direct migration */ ;
3  if With k high-load processing units then
4      for $i = 0$ to $k$ do
5          for $j = 0$ to $h$ do
6              /* $C_i^k$ denotes the emigrated load for the $i^{th}$ processing unit.
                $R_j^h$ maps the data tuples into the $j^{th}$ processing unit */ ;
7              $R_j^h \leftarrow C_i^k$ ;
8          end
9      end
10 end
11 /* Task cutting */ ;
12 /* $LR_j^k$ denote the load of processing unit after migrating task */ ;
13 if $LR_j^k > Lmax$ then
14     Splitting $k1$ into $k11$ and $k12$ ;
15     $R \leftarrow k11, k12$ ;
16     for each $R$ do
17         by slots utilization in descending order $S$;
18         $S \leftarrow R$;
19     end
20 end
21 for each $C_i^j$ do
22     Determine the load information for the next moment $C$;
23 end
24 return Inter-point tuning strategy
```

(b) Inter-node load balancing. For the resource imbalance among Work Nodes, it shows that the load resources of some nodes are in the balanced range, while the resources of other nodes are below the minimum threshold $L_{\min}$ to produce the load imbalance among nodes. Proposes an inter-point tuning strategy for this phenomenon to ensure that the degree of resource allocation among nodes is not significantly different and there are no nodes below the threshold $L_{\min}$.

The resource utilization of each node is sorted and divided into two groups according to the threshold value, and the nodes in the general load group are paired with the nodes in the below-threshold group one by one for migration operations, and polled until all the nodes below the normal range are in the normal range.

(1) The number of low-load nodes is smaller than the number of normal nodes. Pair migration is performed in the following manner: positive order for low-load nodes and backward classification for normal-load nodes.

(2) The number of low-load nodes is greater than the number of normal nodes. Pair migration is performed in the following manner: positive order for low-load nodes and backward classification for normal-load nodes, as shown in Fig. 7.

The low-load nodes that become general load nodes after tuning are added to the general load node queue and continue to participate in pair migration until there are no nodes below $L_{\min}$ in the cluster.

(3) If the pair migration makes the general load node exceed the $L_{\max}$ value called high load node, then it abandons the pairing of the current node and selects the next node in the queue for pairing operation with the current low load node, as shown in Fig. 8.

After intra-node resource migration, the operators within the nodes of the entire cluster are in a relatively load balanced state, and the cluster no longer has nodes above the $L_{\min}$ value. Inter-node load balancing algorithm is described in Algorithm 2. Sort and pair two sets of nodes by step 4. Handle critical nodes by steps 5–10. The tuned pair of nodes is put into the corresponding queue according to the load by steps 12–16, and repeat the above operations until the load is balanced between the nodes. And the above operation is repeated before the load is balanced between the nodes. The time complexity of algorithm 2 is $O(m)$, where m is the number of nodes with their load in normal range.

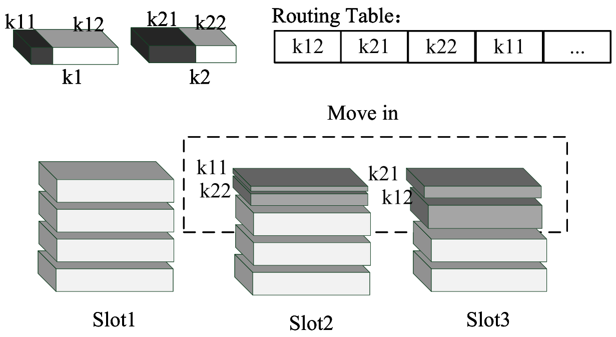


Fig. 6 Splitting key migration into routing table and Move in the key in the routing table

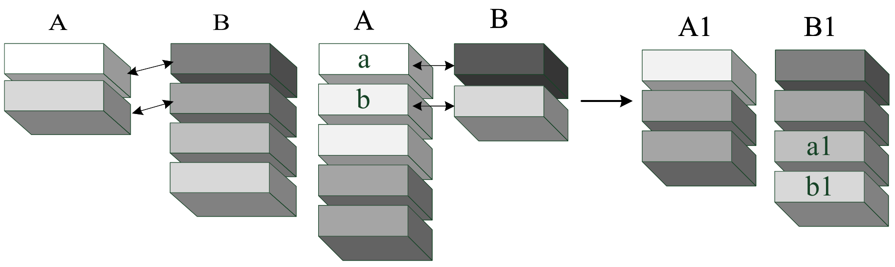


Fig. 7 Point-to-point matching and migration

```
Algorithm 2: Inter-node load balancing algorithm.
   Input: Overloaded compute nodes $Nl_{(1,...,l)}$,Node load within normal
          range $Nm_{(1,...,m)}$;
   Output: Balanced Lightweight Load Resources.
1  $Nl_{(1,...,l)}$ by node load in ascending order ;
2  $Nm_{(1,...,m)}$ by node utilization in descending order;
3  for $j = 1$ to $m$ do
4      //operator pairing with general operator ;
5      $Nm_j \longleftrightarrow Nl_j$
6      if New $Nm_j$ becomes $Nl_j$ then
7          // Migration brings errors;
8          abandon the current node;
9          // Teleport the next person ;
10         Continue;
11     end
12     Migrate the low utilization of slot in $Nl_j$ to $Nm_j$ ;
13 end
14 if numbers $Nm \leq Nl$ then
15     $N_l$ by slots utilization in ascending order;
16     for $k = 1$ to $h$ do
17         $N_{l(1,...,h)} \longleftrightarrow N_{m(i)}$
18     end
19 end
20 return Balanced Lightweight Load Resources.
```

The St-Stream resource allocation algorithm is an arithmetic migration operation within the node according to the degree of load skew, divided into three cases: direct migration c out, direct migration in, and tangential migration in, with a temporary routing table to store the migrated processing units. The above algorithm is repeated in one

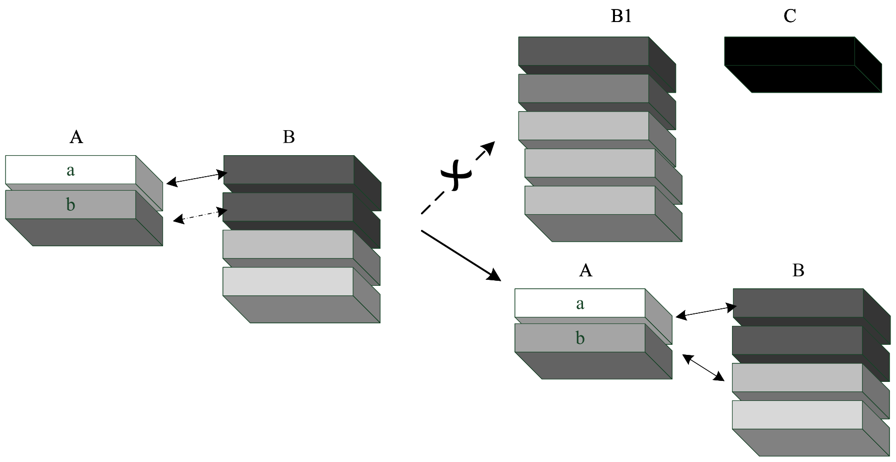


Fig. 8 Recovering erroneous migration operators

clock cycle until the balance is restored from the load skewing case to achieve the effect of load balancing within the node. The Inter-node tuning strategy is used to further perform inter-node tuning for each node after the above load balancing until there are no nodes in the cluster with skew conditions outside the L_{min} range and restore the balance of the cluster. Finally, the load balancing of the whole cluster is achieved.

5 Performance evaluation

The proposed St-Stream system is developed based on Storm 2.2.0, and installed on top of CentOS 7.0, load monitoring using Ganglia Web 3.7.2. The cluster consists of 8 machines, with one machine designated as the master node running storm nimbus and the remaining 7 machines working as supervisor nodes. We use Ganglia to collect the CPU utilization of each node. The software configuration of St-Stream platform is shown in Table 3. The hardware configuration of St-Stream platform is shown in Table 4. In addition, Apache Storm platform is one of the most popular open-source stream computing systems for big data. It has been used by many well-known companies and organizations, such as Twitter and Alibaba, and widely adopted by researchers for comparison purposes [4]. We select Storm as the baseline for testing and analysis.

A wordCount stream application is constructed for experimental testing. The logic graphs of wordCount are shown in Fig. 9. The vertex functions and the number of instances are shown in table 5. In real-world scenarios, skewed data streams occur frequently. For example, a large e-commerce platform needs to compute the consumption statistics of multiple regions in the country. Different

Table 3 Software configuration of the St-Stream

Software	Version	Description
Centos	7.7.1908	Operating system
Apache storm	2.2.0	Stream computing engine
JDK	1.8	Java environment
Apache zookeeper	3.4.6	Collaboration services required by storm
Ganglia	3.7.2	Cluster resource monitoring framework
Redis	2.8.2103	Database
Apache kafka	2.11–2.1.1	Middleware

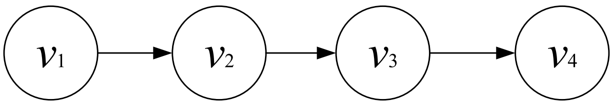
Table 4 Hardware configuration of the St-Stream

Server no.	Configuration information			
	CPU	Memory	Hard disk	NIC
NODE-1,2	Intel(R) Xeon(R) CPU X5650 2.67GHz 4 cores	8GB	80GB	1000Mbps
NODE-3,4	Intel(R) Xeon(R) CPU X5650 2.67GHz 2 cores	4GB	40GB	1000Mbps
NODE-5,6	Intel(R) Xeon(R) CPU X5650 2.67GHz 2 cores	4GB	40GB	1000Mbps
NODE-7,8	Intel(R) Xeon(R) CPU X5650 2.67GHz 2 cores	4GB	40GB	1000Mbps

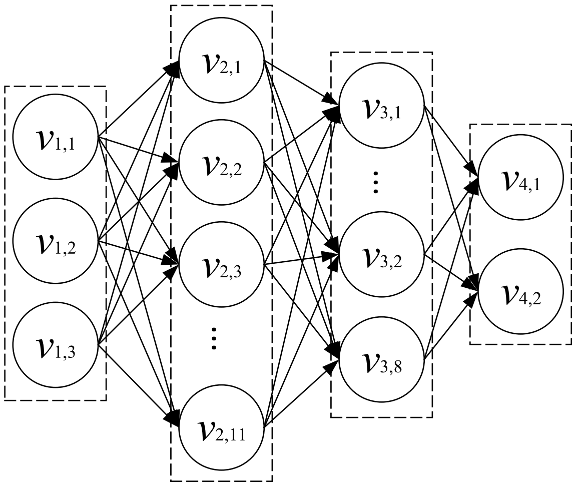
population and consumption levels in each region lead to skewed aggregations of data, which are unbalanced loads on the cluster. To implement a skewed data stream, we partially label the prepared test data and implement a grouping policy based on Apache Storm's built-in interface CustomStreamGrouping. The same labeled data are aggregated to the same nodes, creating an unbalanced load on the cluster. Under the unbalanced load, we then observe three main evaluation criteria including CPU utilization, intra-node slots usage and latency, and the effectiveness of our proposed strategy. The WordCount topology is computation-Intensive, which heavily consumes the CPU resources of compute nodes. On our experimental platform, the storage space of compute nodes is relatively large and memory overload is not observed. Therefore, only the CPU load of nodes is analyzed.

5.1 Inter-node CPU utilization

System CPU utilization reflects the usage of each node, and if the difference in CPU utilization of each node represents the distribution of resources and load within the cluster. The smaller the difference in CPU utilization per node represents a more



(a) Vertices of the topology



(b) Instances of the topology

Fig. 9 Topology of wordCount

Table 5 Vertex functions of Top_N

Vertex	Function
v_1	Read words from Kafka
v_2	Split words
v_3	Count words
v_4	Put the results into the Redis database

even distribution of resources within the cluster and a more balanced load. Send labeled data to seven nodes in the frequency range of (500,3500) and observe the load change of these nodes. Unbalanced load of cluster is achieved because of skewed data stream. Figure 10 shows the CPU utilization of the seven nodes before using St-Stream.

From this figure, we can observe that the CPU utilization of S4 node is between 80% and 90% for a long period of time, which can lead to a long-term skewing of the system load toward S7 node. While the resources of some nodes are idle for a long time, the CPU utilization of S1 node is between 10% and 15% for a long time, which leads to the skewed system load. The CPU utilization of each node is different, with approx. 70% difference between the minimum and maximum.

The use of St-Stream strategy can split a part of the tasks of S7 node to S1 node, S7 is reduced from a long-term high load state to a general load, the free resources within S1 node are utilized and recovered from a super low load state to a general load condition, thus the load of each node gradually recovers from a skewed state to a balanced state and is in the range of load balancing for a long time. In Fig. 11, starting from 240 s, the CPU utilization of each node is between 40% and 60%, and no node's load exceeds 60% or is lower than 40%, and the cluster is in and remains in a relatively balanced state for a long time. St-Stream narrows the gap between nodes' resource loads and tries to stabilize them.

Compared with storm, the CPU resource utilization of each node in St-stream is in a small tilt range, and there is no ultra-high peak or high difference.

As shown in Fig. 12a, it can be observed that the CPU utilization of S1 and S2 are below the minimum threshold (default 30%), triggering a small skewed load in the system. The experimental results show that the CPU utilization of S1 and S2 are 11.25% and 21.2%, respectively, which are less than 30%. However, the CPU utilization of the highest load node S7 in the cluster is about 88.5%, which is 77.25% more

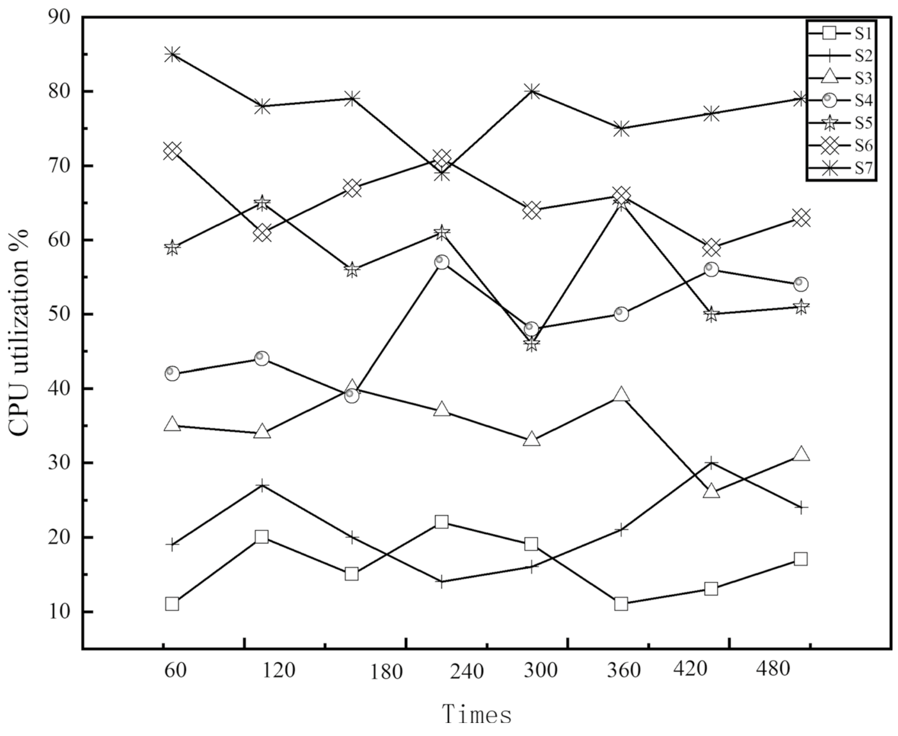


Fig. 10 System CPU utilization curve

than that of S1. After applying the two-tier collaborative load balancing strategy, the loads of S1 and S2 resume to approx. 35.6% and 40%, respectively, both in the normal load range. It can be seen that our proposed strategy can effectively address the load skewing situation generated by lower-loaded nodes. Idle resources are avoided to some extent.

As shown in Fig. 12b, it can be observed that the CPU utilization of S7 is above the maximum threshold, increasing the risk of node downtime. The experimental results show that the CPU utilization of S7 is 88.5%. As the nodes with high resource consumption can affect the response time of system, it is essential to reduce their high CPU consumption. After applying our proposed strategy, the loads of node S6 and S7 are adjusted to approx. 54% and 57%, respectively. At this time, S7 is only 21% more than S1. Our proposed strategy is proved to be effective for dealing with the skewed load.

5.2 Intra-node slots usage

The resource usage of slots within a node reflects the trend of load before and after each node. The more unbalanced the resource allocation is, the larger the difference in actual slots usage is after intra-node operator migration and

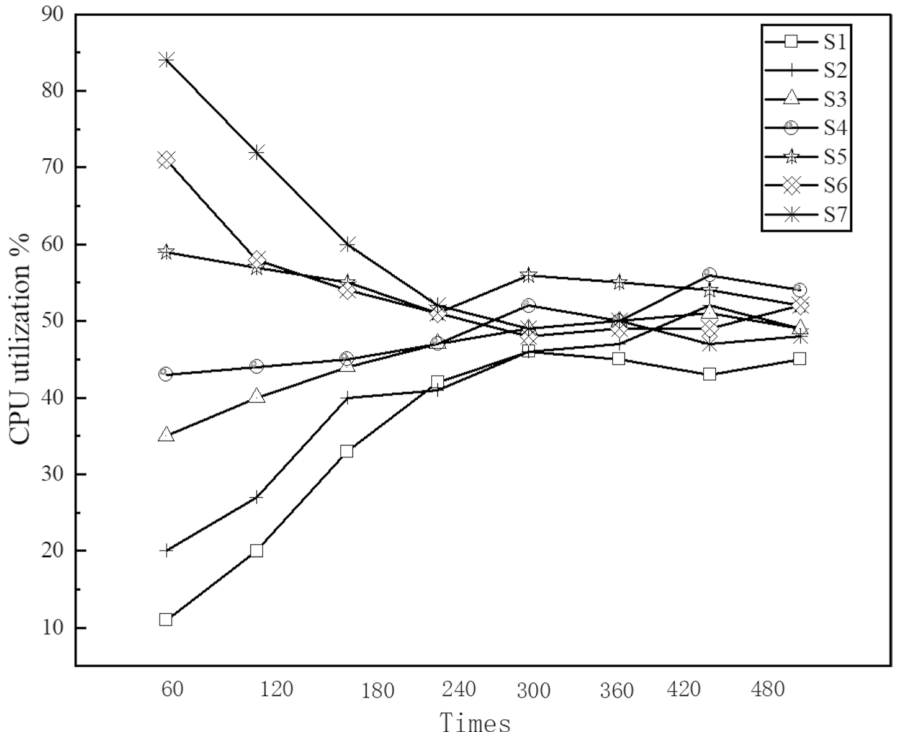
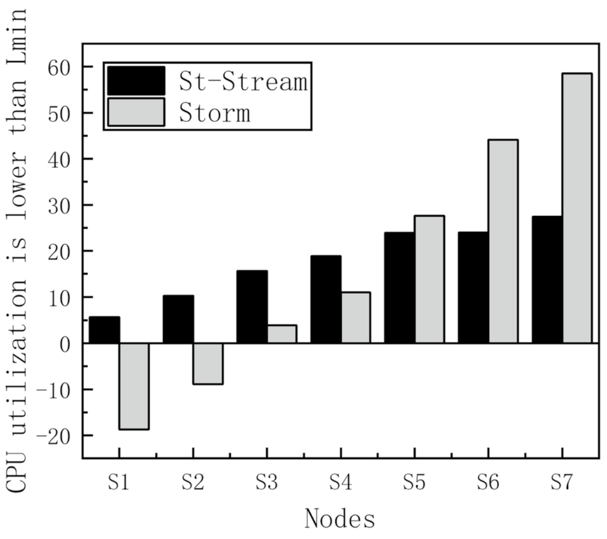
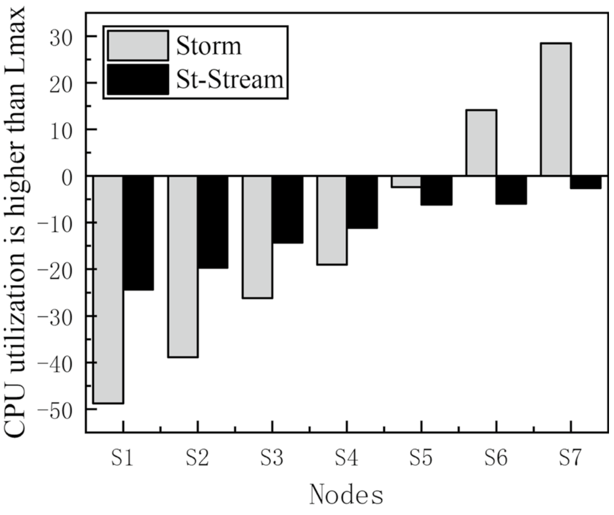


Fig. 11 System CPU utilization after restoring balance



(a) CPU utilization is lower than L_{min}



(b) CPU utilization is higher than L_{max}

Fig. 12 System CPU utilization threshold comparison chart

inter-node load optimization. It is only necessary to check the difference between the before and after slots utilization to know whether the nodes within the cluster can recover from the unbalanced state to balance, and the difference in change is denoted by δ.

As can be seen from Fig. 13, the slots usage of St-Stream is gradually at a stable level compared to Storm. Taking S1 as an example, the difference is 24.375% after using St-Stream, and the point returns from the very low load state to within the general load range, indicating that nearly 24% of the resources are utilized. Taking point S5 as an example, the difference is only 3.75%, saying that the node itself is in a load balanced state and only a very small number of resources are given up. The use of resources in all the nodes in the cluster has improved after using St-Stream. The more unbalanced the load is, the stronger the improvement is.

5.3 System latency

Due to the real-time nature of the data stream, the data gradually decreases in value over time and needs to reflect system performance through latency. The longer the time, the lower the latency, the better the system performance.

From Fig. 14, we can notice that after submitting the topology, the latency of St-Stream is significantly lower compared to Storm, at 200 s the latency of Storm is 17 milliseconds, while the latency of St-Stream is 14 milliseconds, the latency is reduced by 17.6%. St-Stream has a lower response time because it migrates computing resources from the nodes with high resource consumption to the nodes with low resource consumption, in order to maintain a balanced resource load in the system.

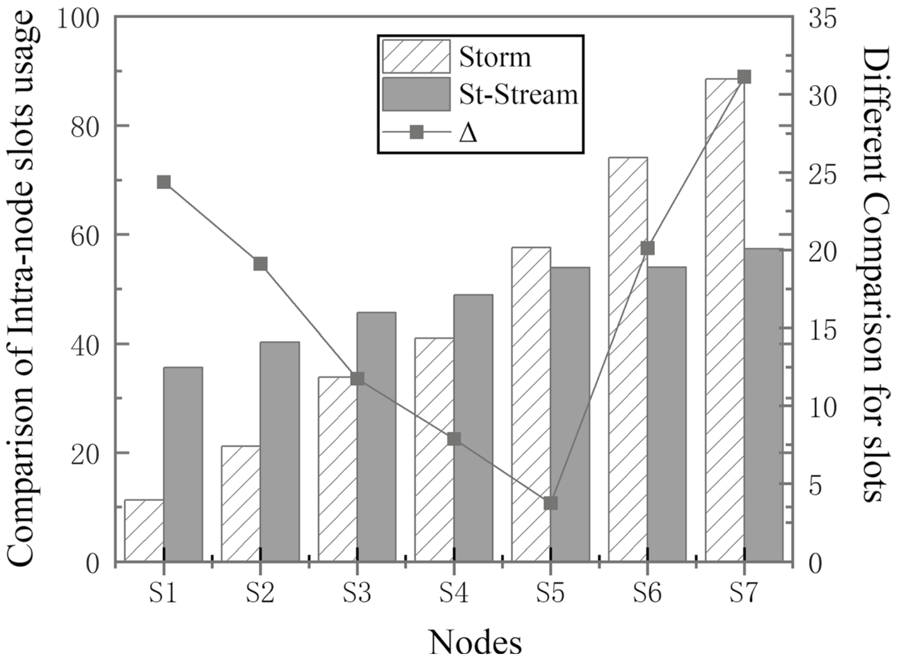


Fig. 13 Before and after comparison of in-point slots usage balance

Therefore, the proposed strategy of balancing the lightweight load resources can better balance the cluster load and improve the cluster performance.

5.4 Load recovery rate

The load skew density distribution curve with axis whisker can visually reflect the distribution pattern of load samples, where the whisker indicates the frequency of data distribution. According to the defined load range, the thinner and higher density curve represents the more concentrated load distribution in a certain part, and the fatter and lower density curve represents the more dispersed load distribution. If multiple density load curves show a scattered distribution, it indicates that there is a skewed load in the cluster and the extremum of loads of these nodes differ greatly. And the loads of nodes need to be adjusted to make the peaks of all density distribution curves concentrate in a small range relatively.

Figure 15 shows the skewed load density distribution curves. The seven nodes in the cluster are scattered in different load ranges and the axes whisker presents the characteristics of distribution coefficients. According to the given load range, it is found that S1 and S2 density curves are in the low load range, and S6 and S7 density curves are out of the specified load range. Therefore, there is a load skew situation between these curves. The density curves with skewed load in Fig. 15 are adjusted

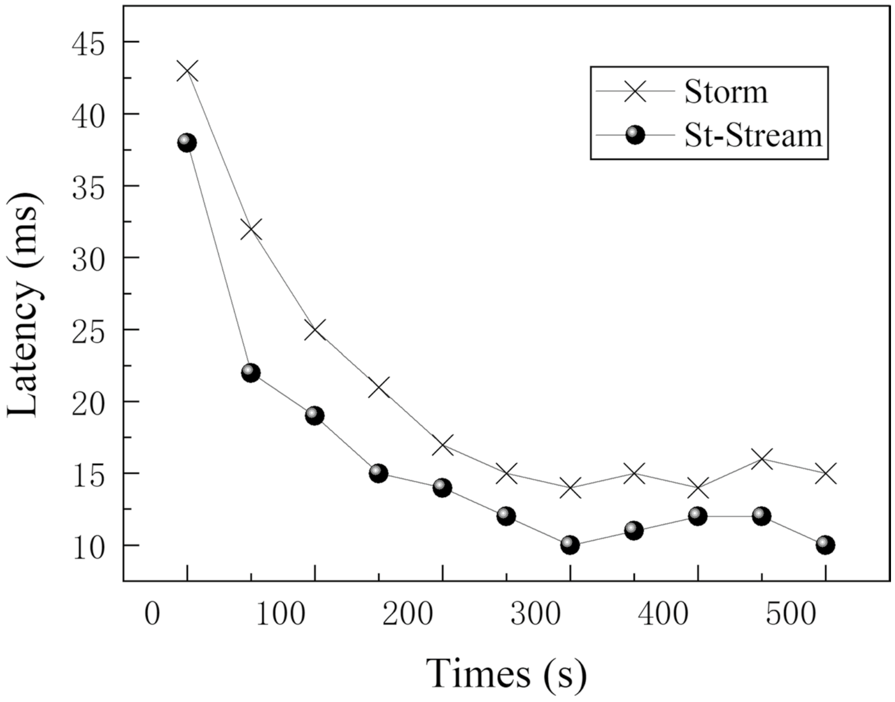


Fig. 14 System Latency balance before and after comparison chart

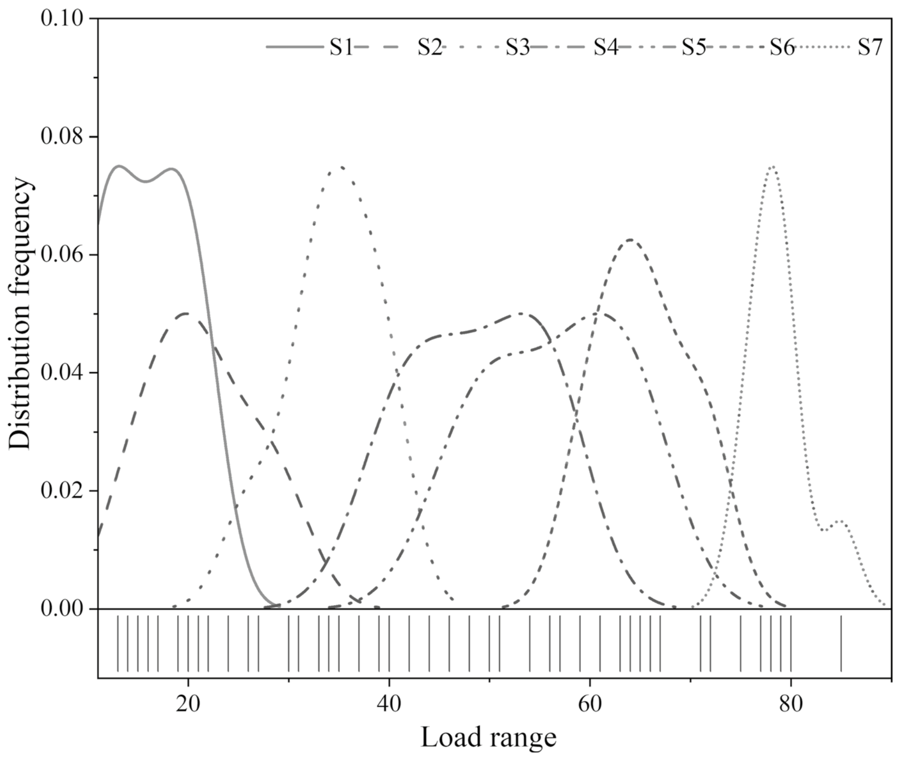


Fig. 15 Skewed load density distribution curves

to make the cluster load recover to a balanced state. After using the two-tier coordination load balancing strategy, the load density distribution curve of the cluster is shown in Fig. 16. In this chart, most of the axial whisker curves are densely distributed in the middle region, and only a few of them are scattered at the two ends. In addition, the peaks of each density curve are relatively concentrated in the load range (40, 60) and exhibit a compact distribution and shows a compact distribution characteristic.

As shown in Fig. 17, the load of all nodes in the cluster is between 0.3 and 0.6 after St-Stream optimization, and the load balance trend line area is stable. From the change of offset of the current cluster, we can see that the load balance degree b is reduced from 1.738 to 1.4401, which means that the current cluster is restored to balance from imbalance, and the load balance restoration rate is 82.86%, compared with Storm, the cluster load is obviously restored to balance from imbalance.

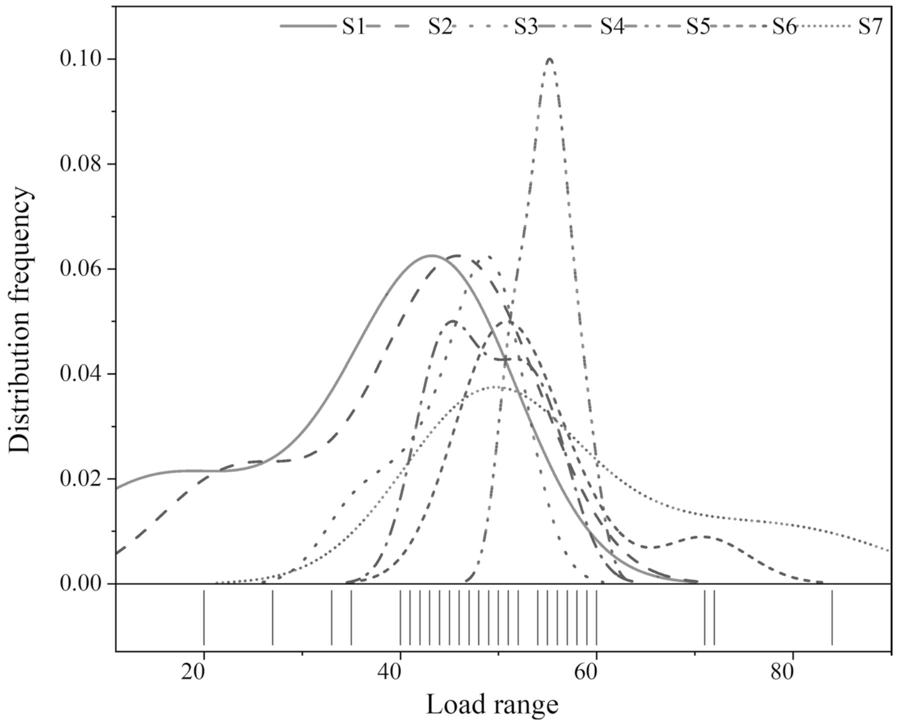


Fig. 16 Balanced load density distribution curves

6 Conclusions and future work

In this paper, based on the study of load balancing methods for existing stream processing systems, we propose a two-tier Coordination load balancing strategy over skewed data Streams. By analyzing the close relationship between the load variation of each node and time after the input data stream of the cluster and the difference of each node's ability to handle the load, perform the corresponding move out and cut migration operations on the slots of overloaded nodes and perform point-to-point optimization for skewed load and resources below the minimum load range so that the load of the cluster can recover from fluctuations to balance. Our experiments and comparative analysis show that the proposed St-Stream strategy can reduce latency, balance cluster load, and improve throughput.

Future work will focus on the following two aspects. (1) Considering the uneven distribution of cluster resources caused by data stream fluctuations in heterogeneous clusters, and ensuring cluster load balancing by classifying and grouping processing according to the different resource allocation methods. (2) We will consider the impact of other aspects such as network, memory, and I/O devices on cluster load.

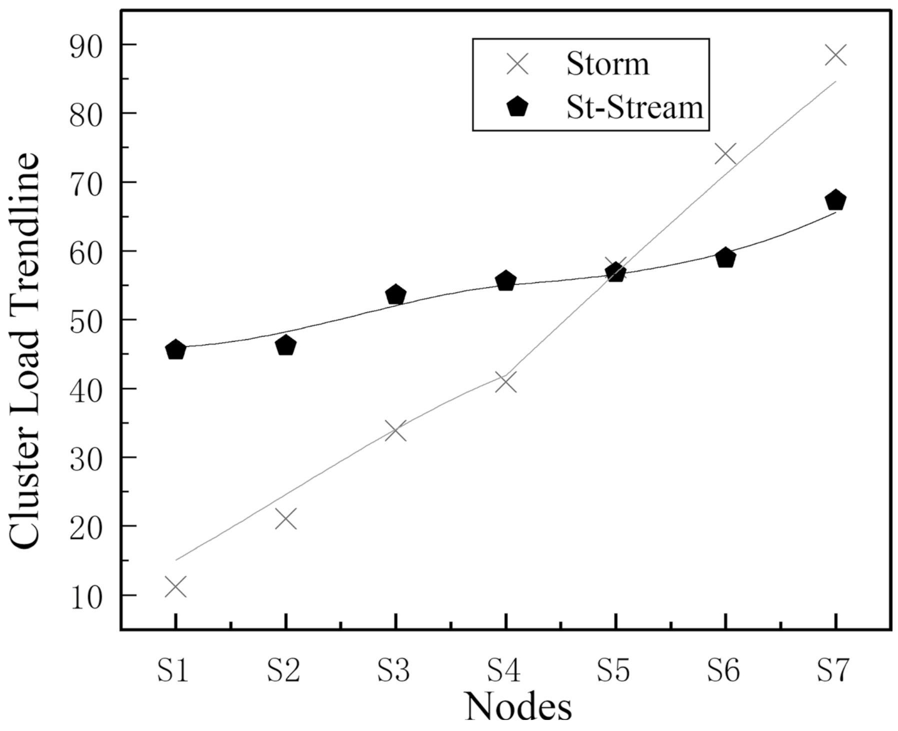


Fig. 17 Cluster Load Trendline

Acknowledgements This work is supported by the National Natural Science Foundation of China under Grant No. 61972364; the Fundamental Research Funds for the Central Universities under Grant No. 265QZ2021001; Melbourne-Chindia Cloud Computing (MC3) Research Network, Australia.

Author contributions DS contributed to methodology, validation, writing–original draft, investigation, and funding acquisition. MW contributed to conceptualization, methodology, validation, and writing–original draft. ZY contributed to validation, investigation, writing–review and editing. AS contributed to formal analysis, investigation, writing–review and editing. RB contributed to methodology, writing–review and editing, supervision, and funding acquisition.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Liu S, Weng J, Wang J, An C, Zhou Y, Wang J (2019) An adaptive online scheme for scheduling and resource enforcement in storm. IEEE/ACM Trans Netw (TON)
2. Baig F, Teng D, Kong J, Wang (2021) Spear: dynamic spatio-temporal query processing over high velocity data streams. 2021 IEEE 37th International Conference on Data Engineering (ICDE), 2279–2284

3. Kumar V, Sharma DK, Mishra VK (2021) Mille cheval: a gpu-based in-memory high-performance computing framework for accelerated processing of big-data streams. J Supercomput 77:6936–6960
4. Aleem M, Islam A (2020) Top-storm: a topology-based resource-aware scheduler for stream processing engine. Cluster Comput J Netw Softw Tools Appl, 123–124
5. Hadian H, Farrokh M, Sharifi M, Jafari A (2023) An elastic and traffic-aware scheduler for distributed data stream processing in heterogeneous clusters. J Supercomput 79:461–498
6. Liu C, Weng J, Wang J, An C, Zhou Y, Wang J (2019) An adaptive online scheme for scheduling and resource enforcement in storm. IEEE/ACM Trans Netw 27:1373–1386
7. Li W, Zhang Z, Shu Y, Liu H, Liu T (2022) Toward optimal operator parallelism for stream processing topology with limited buffers. J Supercomput 78:13276–13297
8. Zhang Z, Jin PQ, Wang XL.(2019) N-storm: efficient thread-level task migration in apache storm. 2019 IEEE 21st International Conference on High Performance Computing and Communication, IEEE 14th International Conference on Smart City, IEEE 2nd International Conference on Data Science and Systems, 1595–1602
9. Qian W, Shen Q, Qin J, Yang D, Yang Y, Wu Z (2016) A slot-aware scheduling strategy for even scheduler in storm. 18th International Conference on High Performance Computing and Communications, 623–630
10. Houatra D, Tseng Y (2018) Monitoring 5g radio access networks with cloud-based stream processing platforms. 2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 1–5
11. Bi Y, Han G, Lin C (2020) Intelligent quality of service aware traffic forwarding for software-defined networking/open shortest path first hybrid industrial internet. IEEE Trans Industr Inf 16:1395–1405
12. Cheng D, Zhou X, Wang Y, Jiang C (2018) Adaptive scheduling parallel jobs with dynamic batching in spark streaming. IEEE Trans Parallel Distrib Syst 29:2672–2685
13. Fischer L, Bernstein A (2015) Workload scheduling in distributed stream proces-sors using graph partitioning. Proceedings of IEEE International Conference on Big Data, Big Data, 124–133
14. Zhao J, Guo J (2018) Design of distance learning streaming media system based on cloud platform. 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), 131–134
15. Shangguan B, Yue P, Wu Z (2017) A stream computing based approach for updating waterlogging information on remote sensing images. 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 373–375
16. Shojaei K, Safi-Esfahani Ayat S (2018) Vmdfs: virtual machine dynamic frequency scaling strategy in cloud computing. J Supercomput 74:5944–5979
17. Fan JHC, Hu F (2015) Adaptive task scheduling in storm. 2015 4th International Conference on Computer Science and Ne The Power of Both Choices Technology (ICCSNT), 309–314
18. Liao X, Huang Y, Zheng L, Jin H (2019) Efficient time-evolving stream processing at scale. IEEE Trans Parallel Distrib Syst 30:2165–2178
19. Jayashri C, Abitha P, Subburaj S, Devi SY, S S, S J (2017) Big data transfers through dynamic and load balanced flow on cloud networks. 2017 Third International Conference on Advances in Electrical, Electronics, In-formation, Communication and Bio-Informatics (AEEICB), 342–346
20. Deng S et al (2020) Dynamical resource allocation in edge for trustable internet-of-things systems: a reinforcement learning method. IEEE Trans Industr Inf 16:6103–6113
21. Grandl R, Chowdhury M, Akella A (2016) Altruistic scheduling in multi-resource clusters. Proceedings of OSDI'16: 12th USENIX Symposium on Operating Systems Design and Implementation, 65–80
22. Son SHI, Moon YS (2021) Stochastic distributed data stream partitioning using task locality: design, implementation, and optimization. J Supercomput **10**
23. Aslam, Adeel HC, H J (2021) Pre-filtering based summarization for data partitioning in distributed stream processing. Concurr Comput Pract Exp
24. Li W, Liu D, Chen K, Li K, Qi H (2021) Hone: mitigating stragglers in distributed stream processing with tuple scheduling. IEEE Trans Parall Distrib Syst, 99
25. FeiChen SongWu HaiJin (2018) Network-aware grouping in distributed stream processing systems. In: International Conference on Algorithms and Architectures for Parallel Processing
26. Qian W, Shen Q, Qin J, Yang D, Yang Y, Wu Z (2016) S-storm: a slot-aware scheduling strategy for even scheduler in storm. 2016 IEEE 2nd Interna-tional Conference on Data Science and Systems, HPCC, Sydney, NSW, Australia, 623–630

27. Nasir MAU, Morales G, García-Soriano D, Kourtellis N, Serafini M (2015) The power of both choices: Practical load balancing for distributed tream processing engines. 2015 IEEE 31st International Conference on Data Engineering, 137–148
28. Sun D, Yan H, Gao S, Liu X, Buyya R (2018) Rethinking elastic online scheduling of big data streaming applications over high-velocity continuous data streams. J Supercomput 74:615–636
29. Lang K, Chai X (2022) Implementation of load balancing algorithm based on flink cluster, pp 264–268
30. Dai Q, Qin G, Li J, Zhao J, Cai J (2023) A resource occupancy ratio-oriented load balancing task scheduling mechanism for flink. J Intell Fuzzy Syst 44:2703–2713
31. Li Z, Yu J, Wang Y, Bian C, Pu Y, Zhang Y, Liu Y (2020) Load prediction based elastic resource scheduling strategy in flink. J Commun 41:92–108
32. Anis Uddin Nasir M, G, DFM, Kourtellis N, Serafini M (2016) When two choices are not enough: balancing at scale in distributed stream processing. 2016 IEEE 32nd International Conference on Data Engineering, ICDE, Helsinki, Finland, 589–600
33. Chen H, Zhang F, Jin H (2021) Pstream: a popularity-aware differentiated distributed stream processing system. IEEE Trans Comput 70:1582–1597
34. Aslam A, Chen H, Jin H (2021) Pre-filtering based summarization for data partitioning in distributed stream processing. Concurr Comput Pract Exp **33**
35. Fu TZJ, Ding J, Ma RTB, Winslett M, Yang Y, Zhang Z (2015) Drs: dynamic resource scheduling for real-time analytics over fast streams. 2015 IEEE 35th International Conference on Distributed Computing Systems, Colum-bus, OH, USA, 411–420
36. Zhang W, Duan P, Gong W, Lu Q, Yang S (2016) A load-aware pluggable cloud strategy for real-time video processing. IEEE Trans Industr Inf 12:2166–2176
37. Cardellini V, Grassi V, Presti FL, Nardelli M (2015) Poster: distributed qos-aware scheduling in storm". Acm International Conference on Distributed Event-based Systems, 344–347
38. Zhou Y, Liu Y, Zhang C, Peng X (2020) Toss: a topology-based scheduler for storm c1usters. IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW, 587–588
39. Wu M, Sun D, Cui Y, Gao S, Liu X, Buyya R (2022) A state lossless scheduling strategy in distributed stream computing systems. J Netw Comput Appl 206:1–16
40. Vicentini C, Santin A, Viegas E, Abreu V (2019) Sdn-based and multitenant-aware resource provisioning mechanism for cloud-based big data streaming. J Netw Comput Appl 126:133–149
41. Fischer L, Bernstein A (2015) Workload scheduling in distributed stream processors using graph partitioning proceedings. IEEE International Conference on Big Data, Big Data., 124–133
42. Li B, Zhang Z, Zheng T, Zhong Q, Huang Q, Cheng X (2020) Marabunta: continuous distributed processing of skewed streams. 2020 20th IEEE/ACM Inter-national Symposium on Cluster, Cloud and Internet Computing (CCGRID), 252–261