Received: 9 May 2023 | Revised: 20 October 2023 | Accepted: 6 March 2024

DOI: 10.1002/cpe.8085

RESEARCH ARTICLE

WILEY

# Lc-Stream: An elastic scheduling strategy with latency constraints in geo-distributed stream computing environments

Dawei Sun[1] | Yueru Wang[1] | Jialiang Sui[1] | Shang Gao[2] | Jia Rong[3] | Rajkumar Buyya[4]

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

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

[3]Department of Data Science & AI, Faculty of IT, Monash University, Melbourne, Victoria, Australia

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

**Correspondence**
Dawei Sun, School of Information Engineering, China University of Geosciences, Beijing, 100083 People's Republic of China.
Email: sundaweicn@cugb.edu.cn

**Funding information**
National Natural Science Foundation of China, Grant/Award Number: 62372419; Fundamental Research Funds for the Central Universities, Grant/Award Number: 265QZ2021001; Australian Research Council (ARC) Discovery Project

## Summary

An effective scheduling strategy is critical for achieving better performance in real-time stream processing systems. How to quickly and efficiently process real-time data stream is always challenging, especially when clusters are collaborating in a Geo-Distributed computing environment. To address these challenges, we propose an elastic scheduling strategy with Latency Constraints in Geo-Distributed stream computing environments called Lc-Stream. This article discusses our work from the following aspects: (1) An optimized data stream redirection method that is proposed based on queuing network algorithm, along with a computing resource model, a latency constrained scheduling model and a communication energy consumption model. (2) An updated node selection method based on the inter-layer task correlation, to reduce the communication latency between groups at the executor granularity. (3) A network cluster distribution for Geo-Distributed computing environment to ensure energy saving under low transmission latency. Experimental results show that compared to R-Storm, Lc-Stream reduces total latency by over 19% and increases throughput by over 37% in typical cross-domain multi-task topologies. Compared to Ts-Stream, Lc-Stream also reduces total latency by over 15% and increases throughput by over 21%. At the same time, it helps to balance the load among the systems and avoid overuse of compute nodes.

**KEYWORDS**

geo-distributed stream computing, latency constraints, load balancing, resource scheduling, storm

## 1 | INTRODUCTION

The era of big data brings us rich data resources and challenges to big-data processing, particularly to large-scale stream data processing. Many stream data processing tasks such as financial investment and commodity trading have specific requirements to obtain the results in a real-time manner.[1] These requests make the real-time streaming systems widespread adopted by many big companies like Twitter, Google, Amazon, Groupon and Netflix.[2]

Unlike traditional data processing systems that store and process static data periodically, streaming processing systems are expected to process data when resources are available. Batch data processing is one of the main methods to process big data streams in the early stage due to its efficiency. The data are collected beforehand and processed by programs at once. That is, the batch data processing is not real-time or near real-time. Another issue is that latency is generated in the data processing phase, often causing partial results that may not be useful in many applications. To solve these problems, real-time streaming systems usually adapt efficient batch-orientated approaches and monitor data items as they arrive simultaneously.

  

The performance of a real-time streaming system is measured by the logical correctness of the operations and whether the result is generated on time.[3] The system needs to complete all the required operations within a specific pre-determined time slot responding to external or internal, synchronous or asynchronous communication. If the time constraints are not met, the system error still occurs. Thus, the real-time streaming systems can be used to identify and process discrete events within a pre-defined time range.

Common big data real-time stream computing systems such as Storm,[4] Spark Streaming[5] and Flink[6] play an important role in many fields, including online system monitoring,[7] mobile sensing data processing[8] and the Internet of Things,[9] financial risk control,[10] recommendation systems,[11] and threat detection.[12]

Storm is a relatively low latency real-time distributed stream processing framework that comes with a default scheduling scheme. Storm uses a pooling method in its allocation process, which often causes load skew.[13] Besides, uneven resource allocation could increase latency and reduce throughput, resulting in low system efficiency.[14] At the same time, the waiting time caused by the communication between nodes is also easily ignored. There is no single system that could perfectly solve these problems so far.

At the same time, due to the widespread distribution and diverse geographical locations of data production platforms, geo-distributed collaboration is particularly important in data processing. Communication latency within geo-distributed clusters, especially in larger clusters, and their energy consumption lead to a certain degree of waste, which directly or indirectly affects efficiency. Therefore, our research aims to provide a reasonable and effective optimization model and scheduling strategy in geographically distributed environments to ensure low latency, low energy consumption and high throughput for the system.

Motivated by the above issues and problems, an elastic scheduling strategy with Latency Constraints (Lc-Stream) is proposed. It aims to improve the efficiency of a real-time steaming system from the following three aspects:

- An optimized data stream redirection method that is proposed based on the queuing network algorithm, as well as a computing resource model, a latency constrained scheduling model and a communication energy consumption model.
- A modified node selection method based on the relevance of the task, to reduce the communication latency between groups at the executor granularity.
- A network cluster distribution constructed in a Geo-Distributed streaming computing environment, to measure energy saving schemes and ensure low transmission latency.

The rest of the article is organized as follows: Section 2 explains the Geo-Distributed environment and the related work in resource scheduling study. Section 3 identifies three main problems addressed in this work. Section 4 introduces the proposed Lc-Stream framework and the corresponding optimization algorithms. Section 5 presents the experiment settings and evaluation results. Section 6 concludes the entire article with future work.

## 2 | RELATED WORK

In this section, we summarize two broad categories of related work: Geo-Distributed environment and resource scheduling in stream processing.

### 2.1 | Geo-distributed environment

Big data processing applications require the analysis of large amounts of data generated in a geographically distributed manner.[15] For example, the geographic distribution of social network users is very wide, often requiring the analysis of massive data generated rapidly across multiple data centers, in order to provide reliable and low-latency services (such as interest-based recommendations) to users.[16]

A Geo-Distributed environment refers to a technical architecture that uses computers and communication technologies to connect computer systems located in different geographical locations, in order to achieve functions such as data sharing, collaborative work, and service provision. It usually consists of multiple nodes scattered across different geographic locations, communicating and collaborating through networks. Geo-Distributed clusters support hybrid and multi-cloud scenarios, and improve high availability beyond single-cluster multi-region deployment.[17]

Geo-distributed clusters are usually considered for remote deployment.[18] Proper deployment and resource scheduling can effectively improve system efficiency. In practical scenarios, due to security and cost considerations, some services and data cannot be placed on all physical machines. Intensive communication and collaboration are required between nodes. In summary, geographically distributed clusters are a powerful computing architecture that can meet the needs of users well.

In recent research, the primary focus of scheduling studies in geo-distributed environments has been to achieve low cost, low latency, and high throughput. Hu et al.[19] proposed a new task scheduling algorithm called Flutter, which considers both network budget constraints and data center

geographic distribution to schedule tasks closer to the data. This reduces task processing time, network traffic, and related network costs. However, there is still room for load balancing improvements. Li et al.[20] developed an M/M/C queuing theory model to address the issue of job scheduling by considering the idle and busy states of the cloud. They derived optimal job arrival rates using Lagrange multipliers. This approach achieved load balancing in job scheduling, reduced response time, and improved throughput. However, they did not take into account the network cost. Two years later, Li et al.[21] proposed a new data placement strategy based on dynamic RDD weights in Spark. In the case of multi-task scheduling with multiple nodes, nodes with strong computing power are selected to place tasks. Li et al.[22] introduced a Lagrange Relaxation method for the data placement problem in geographically distributed environments. This method comprehensively considers multiple factors to minimize data transmission time. They also designed a fault-tolerant scheduling model to selects the best nodes by considering task execution time and energy consumption.

Russo[23] proposed a framework for organizing adaptation using a hierarchical control approach within geographically distributed infrastructure. Their approach involves a decentralized strategy based on reinforcement learning for elastic scaling in geographically distributed environments. This work also considers heterogeneous environments. Chen et al.[24] proposed a transformation-based software allocation algorithm with the goal of minimizing cost, taking into account the characteristics and price heterogeneity of software among geographically distributed data centers. This algorithm can improve cost efficiency and reduce communication.

Geo-distributed cloud computing is slightly different from streaming computing. The studies mentioned above are closely related to stream processing, but there is limited research that combines geo-distributed environments with stream computing. This article bridges the gap by merging geographically distributed environments with stream processing to refine the scheduling problem.

## 2.2 | Resource scheduling in stream processing

Improving the efficiency of large-scale and bound computing in current stream computing environments is a popular topic in the last decade. Through a parallel computing system composed of multiple processors (i.e. multi-core processor cluster), an effective scheduling strategy can improve system efficiency and reduce resource waste. While increasing the overall phase rate of the system, which refers to the rate at which the system as a whole processes tasks or computations, issues such as whether the load of each part of the system is skewed or whether the communication overhead between groups is too large remain unresolved. The default scheduler uses a polling allocation method, which means task allocation is relatively random. When planning worker resources, it must be allocated to the maximum extent that may be required during the runtime of each worker. If the content of a single worker is large, operations such as heap dump may cause the worker busy for a long time. In extreme cases, if the worker is time out, then the supervisor kills that worker immediately.

Many efforts have been put to find out the best resource scheduling methods in the past decade. In 2013, Aniello, Baldoni and Querzoni[25] proposed two advanced general-purpose schedulers for Storm to improve the performance of various topologies. The first one is a static scheduler. It works offline by analyzing the topology and adjusting the deployment. The second one is a dynamic scheduler. It improves performance by continuously monitoring system performance and rescheduling the deployment during runtime. A monitoring module is added to collect the Tuples transmission rate, judge the load level of executors based on the rate value, and then sort them in descending order. One year later, a new storm framework with a load monitoring module and an optimized scheduler was proposed to store the collected information in database, called T-Storm.[26] The core component of the system is the traffic-aware online scheduling algorithm. Specific conditions can be set. Once the conditions are met, the Executor is assigned to the corresponding Slot, and the traffic-aware online scheduling is used to speed up data computing. In addition, T-storm supports hot-swap scheduling algorithms and timely adjustment of scheduling parameters, which is transparent to users.

R-storm was proposed to provide a resource-constrained scheduler.[27] It resources (memory, bandwidth and CPU) for mathematical modeling, abstract into a three-dimensional space. The multi-dimensional resources are used to improve the accuracy of scheduling decisions. The breadth-first search algorithm is used to traverse and sort the components. Although R-storm performs better than the default scheduling method, it cannot control the performance when CPU sharing occurs.[28] Meanwhile, Fu's team proposed a dynamic resource scheduler for real-time analytics over fast streams based on the current workload to avoid wasting resources or failing to deliver correct results on time.[29] Wang et al. [30] further expanded their work to address three challenges: modeling the relationship between allocated resources and query response time, finding the best placement of resources, and controlling the overhead of measuring system load to a minimum. The model uses Er-lang and Jackson network models and effectively analyzes and summarizes each operator.

Recently, Tantalaki et al.[31] developed a pipeline-based linear scheduler that handles one-to-many allocation through a communication refinement algorithm and a mechanism. They use a general topology-aware formula for matrix transformation task assignment and scheduling problems. Ts-Stream[32] is also a topology-aware scheduling strategy. It consists two modules: the first module uses Laplacian to partition the topology graph, while the second calculates the node selection method. Sun et al.[33] proposed an energy efficient and runtime-aware framework. They modeled the stream application, resources, and energy consumption, formalizing the scheduling problem. But they didn't take into account communication between the data. Eskandari et al.[34] proposed a heuristic scheduling method to effectively reduce the communication cost by fully utilizing and consolidating cluster resources. Hadian et al.[35] investigated the problem of elasticity and scaling decisions and proposed a method named ER-Storm to

**TABLE 1** Comparison of between Lc-Stream and related work.

| | Related work | | | | | | |
|---|---|---|---|---|---|---|---|
| **Parameter** | **32** | **33** | **34** | **35** | **36** | **37** | **Lc-Stream** |
| Latency constrained | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Status management | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Border strategy | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| Prediction strategy | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ |

**TABLE 2** Description of main symbols used in the Lc-Stream models.

| Symbol | Description |
|---|---|
| $\lambda_n$ | Arrival rate |
| $\lambda_0$ | Initial arrival rate |
| $\mu_n$ | Processor processing rate |
| $\mu_0$ | Initial processing rate |
| $\lambda_e$ | Effective arrival rate (number of tuples entering executor per unit time) |
| $Corr$ | Distance-latency correlation |
| $Gd$ | Fround distance |
| $V_{Gd}$ | Distance variance |
| $\rho$ | System load |
| $p_n$ | System steady state distribution |
| $p_0$ | System initial state distribution |
| $L_k$ | Average queue length of worker node |
| $W_k$ | Average waiting time of the data tuple |
| $T_{latency}$ | Communication latency |
| $V_{latency}$ | Latency variance |

suppress the communication overhead and select the appropriate worker node to host the relocated operator. Farrokh et al.[36] proposed a scheduling algorithm for SP-Ant, which uses ACO algorithm to find the desired assignment of working nodes by the executor by considering local optimization and following the global optimization strategy at the same time, so as to find the optimal assignment of operators. This method also applies to heterogeneous clusters. Zhang et al.[37] proposed a prediction method to address the problem of data stream fluctuations by forecasting the trend of data stream fluctuations in the next time window for cluster optimization of resource allocation. Unfortunately, using deep learning models for prediction introduces additional overhead. In order to achieve fair network usage and reduce latency, the scheduling method proposed by Salem et al.[38] runs on a group of senders and receivers, and channels the distribution of large-capacity data streams with high throughput, so that streams from the same observation time are quickly gathered in the compute nodes.

To summarize, the above studies provide valuable insight into the challenges and potential solutions for scheduling problem in distributed systems. Besides, the comparative parameters of the research should be improved to have a reasonable allocation of resources, to achieve low latency and high through put with high CPU availability and workload, and to provide suitable measurements for different scheduling requirements. The summary of the comparison between our work and the closely related work is given in Table 1.

## 3 | PROBLEM STATEMENT

Three main problems are particularly considered in this work: (1) the division of computing resources, (2) latency constrained, and (3) the communication model of cross-regional clusters. This section elaborates these three issues.

We summarize the notations used in this article and their descriptions in Table 2.

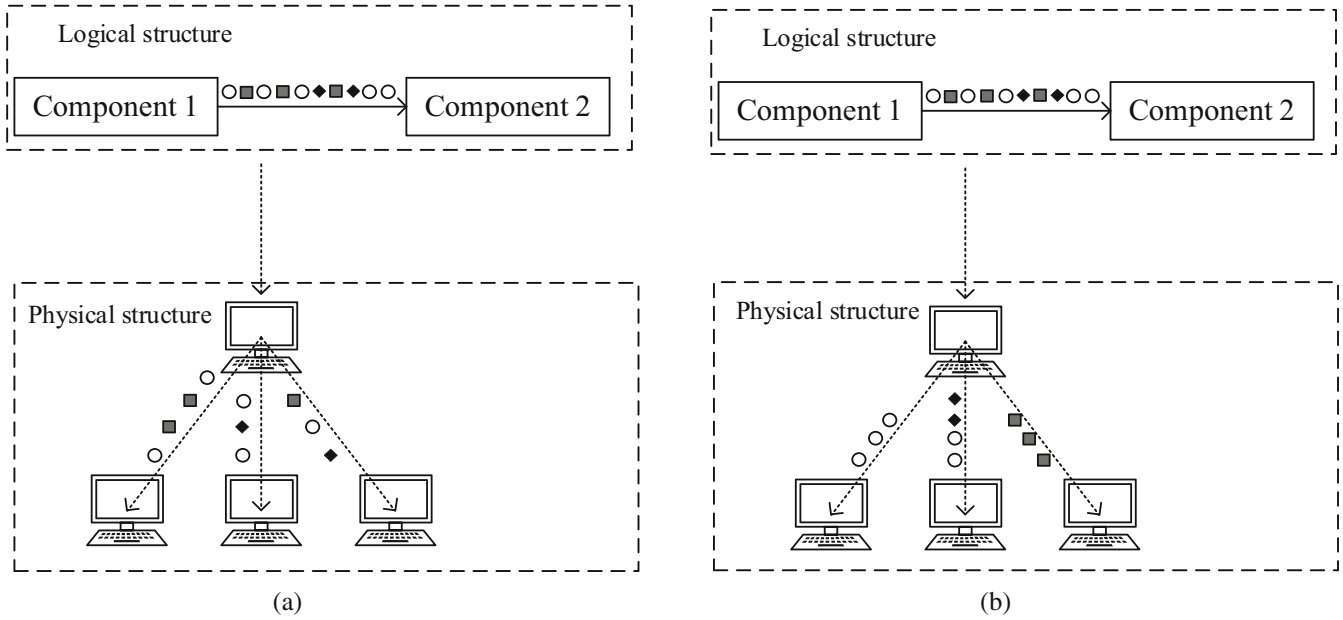


**FIGURE 1** Different ways of assigning tasks.

## 3.1 | Computing resource model

When modeling real-time data computing, it is very common to mix a large amount of data and complex computing. Because of the characteristics of real-time data streams and irregular reception and processing, it is hard to understand the scale of date and the number of resources required.[39] It results in resource shortage and data stream congestion, and slow processing speed. Resource allocation with hindsight will naturally consume a lot of unnecessary time, thereby affecting the system's overall performance. For systems that require real-time feedback, low latency becomes a key issue.

At present, most task allocation models are basically based on two schemes: allocating tasks according to the availability of worker node resources, or create optimal subgraphs according to the correlation of task's topological graph for node selection. Our idea is to design and implement a new algorithm for task selection. A task selection method with higher fault tolerance should be able to monitor the resource occupancy of one node through a monitoring module. The communication between tasks includes related tasks with more intensive communication and unrelated tasks with less or no communication. The task identification id is set to distinguish whether they are related. When the worker node's resources are sufficient, related tasks are placed on the same worker node for processing, otherwise the node with the most resources will be selected to take over. The maximum resource usage limit can be set to ensure communication latency while avoiding overloading a single worker node.

As shown in Figure 1A, Storm's default scheduling mechanism uses polling to schedule and allocate tasks. It distributes tasks equally to each worker node. This distribution does not consider latency optimization. The assignment of related tasks in upstream and downstream components to different worker nodes will bring communication pressure to the system. This is an important reason for the high cost of communication between nodes and processes. The ideal method to assign tasks is to place related tasks on the same worker node.

Figure 1B, it is an ideal scheduling task distribution strategy. It divides tasks according to their relevance. Associated tasks are processed in the same worker node. In the case of limited resources, if tasks can be allocated according to the available resources of each physical machine, the higher availability of CPU of each node and the overall load balance of the system can be achieved. Assigning tasks to the same worker node can also optimize the communication cost between nodes.

The problem is how to manage the resources of each node in a unified manner, so that tasks can be allocated properly according to the resource consumption condition.

## 3.2 | Latency constrained scheduling model

The second model we consider is the Latency constrained scheduling model. It is an improved queuing network model. The queuing network model is a mathematical model obtained by quantifying real-time passenger stream and the number of service stations. By analyzing the relationship between customer arrival rate and service rate, we can get the best service strategy.[40] We change it into a latency constrained scheduling model to

solve the scheduling problem for real-time data streams. The model is applied by treating passenger stream as data stream and service station as executor. It makes it more suitable for the operation of the computing system, so as to obtain a more reasonable scheduling strategy.

Assume that the arrival rate and processing rate of an executor can be calculated by (1) and (2).

$$\lambda_n = \frac{\lambda_0}{(n+1)^a}, n = 0, 1, 2 \ldots , \tag{1}$$

$\lambda_n$ is the arrival rate. $\lambda_0$ is the initial arrival rate. $a$ is fixed value which is related to the actual model initialization state and usually satisfy $a < 0$.

$$\mu_n = \mu^b \mu_1, n = 1, 2 \ldots , \tag{2}$$

$b$ is fixed values which is related to the actual model initialization state and usually satisfy $b > 0$. $\mu_n$ is the processor processing rate. $\mu_0$is the initial processing rate. Consider the processing model of each executor: $\rho$, $p_n$,$p_0$ can be calculated by (3),(4) and (5).

$$\rho = \frac{\lambda}{\mu} < 1, \tag{3}$$

$$p_n = \frac{\rho_n}{n!} p_0, n = 1, 2, \ldots , \tag{4}$$

$$p_0 = e^{-\rho}, \tag{5}$$

where $\rho$, $p_n$ and $p_0$ represent the system load, the system steady state distribution and the system initial state distribution of the system, respectively. The ratio of the arrival rate $\lambda$ to the processing rate $\mu$ is the system load $\rho$. The average queue length of worker node $L_k$ can be calculated by (6).

$$L_k = \sum_{n=0}^{\infty} n p_n = \sum_{n=0}^{\infty} \frac{n \rho_n}{n!} p_0 = \rho, \tag{6}$$

$p_n$ is calculated by (4), $p_0$ is calculated by (5), and $\rho$ is the result of final simplification. Effective arrival rate $\lambda_e$ (the number of tuples entering the executor per unit time) can be calculated by (7).

$$\lambda_e = \sum_{n=0}^{\infty} \frac{\lambda}{n+1} p_n = \mu(1 - e^{-\rho}). \tag{7}$$

The average waiting time of the data tuple $W_k$ can be calculated by (8).

$$W_k = \frac{L_k}{\lambda_e} = \frac{\rho}{\mu(1 - e^{-\rho})}. \tag{8}$$

The above equations show a basic dependency model for a single worker node. It provides the arrival rate $\lambda_n$ and processor processing rate $\mu_n$ of a topological tuple on a single worker node. Among them, Formula (6), (7) and (8) are important parameter indexes in Lc-stream model. The current load and steady-state distribution of the system are obtained from the arrival rate and processing rate. And then calculate the average queue length $L_k$, for system executor quantity proportion adjustment index. Because the processing capacity of a single node is limited, so need to compute the length of the tuple to be processed per unit time and the effective arrival rate $\lambda_e$. $\lambda_e$ as another important reference index, can derive the average waiting time $W_k$.

After determining the queuing network model for a single worker node, we add a complex model for cluster scheduling. The entire cluster includes multiple worker nodes. Through what we do on a single worker node, the best or lowest processing latency range can be obtained. A monitoring module is introduced to track the tuple arrival rate and processing rate for each worker node. The monitoring module monitors node resource usage, allocates resources, and prevents worker nodes from being overloaded. Therefore, it can control the allocation of each worker node and the ideal latency constraint through the allocation of node resources and the assignment of tasks to nodes. In this way, the latency of the entire system can be optimized.

## 3.3 | Energy consumption model

Due to different origins of data streams, it is common to have the system run in a cross-regional cluster mode. When considering latency, the communication latency between networks should not be ignored. As network is a more common factor that influences the communication latency, we

mainly consider the high-latency issues caused by the server distance and network congestion. Our aim is to effectively and reasonably plan the low-latency and high-throughput task processing and scheduling scheme for cross-regional clusters.

Given that the correlation between distance and latency is a first-order linear relationship, the transmission latency between different hosts is measured by the smallest sending and response time of data packet. It is represented by (9):

$$T = t(i,j) \Leftrightarrow t(i,j) = \min\left(t_1 + t_2 + \cdots + t_j\right), \tag{9}$$

where $t_1 + t_2 + \cdots + t_j$ are the sending and response time of data packet. $T$ is the transmission latency. Where $i$ and $j$ indicate adjacent server nodes.

In consideration of communication latency, we mainly control energy consumption by selecting server nodes and servers processes on the server nodes. The ultimate goal is to ensure normal work and load balance with sufficient resources, and use the least number of nodes. To enhance the correlation of distance latency, we introduce the classic definition of correlation between latency variance and distance variance. In the article, we improve the basic distance-latency correlation formula and derive a latency correlation expression method for the entire network topology. The energy consumption is directly related to the distance latency of the entire network topology. Therefore, the energy consumption is indirectly measured through the correlation. The metric can be expressed by (10):

$$Corr = \frac{\operatorname{cov}\left(T_{latency}, Gd\right)}{\sqrt{V_{latency} + V_{Gd}}}, \tag{10}$$

where $Corr$ is the distance-latency correlation, $T_{latency}$ is the communication latency, $Gd$ is the ground distance. $\sqrt{V_{latency} + V_{Gd}}$ and $V_{Gd}$ are the latency variance and the distance variance, respectively.

In our real-world environment, there are more than 3 servers in each resource center, making it more helpful to study the latency of a center and the latency of these far apart resource centers. While aiming at reducing latency and improving efficiency, energy consumption can also be optimized accordingly. As is an online real-time processing platform, Storm's long-term service support inevitably puts pressure on energy consumption. When the resources are sufficient to meet the processing requirements, we restrict the energy consuming resources. As the energy consumption costs generated by different distant nodes are also different, in our designed scheme, unnecessary waste of energy costs is avoided to the best extent. To choose an appropriate regional node and a reasonable number of regional node servers, we focus on the priority and judgment conditions of regional nodes.

As the communication cost influenced by bandwidth might be unstable, we use Round-Trip-Delay-round-trip (RTD) latency to represent the communication latency between remote nodes. To communicate with the target node, connection is to be established in the same time interval as sending a request. After the target node receives the request, it immediately responds to estimate the RTD of the pre-established connection. RTD latency consolidates the time series list, sorts Physical machine in each resource center in descending order according to the response time, judges the latency of the unused area nodes at that moment, and establishes connections with the first nodes in the time series table.

In terms of energy consumption, we perform simple data packet merging to save energy. A server with available resources and low network communication latency receives tuples after the merge. Our data package contains task stream, node id and valid data. After the task streams are merged, an appropriate worker node is selected through Lc-Stream scheduling. Only the servers participating in the topology processing will be considered for energy consumption. We choose as few servers as possible for the same topology processing, and make predictions about the worker node and number of server nodes to reduce the scheduling time. In general, when the CPU resources are sufficient and satisfy the condition of load balancing, energy consumption can be lowered.

In order to obtain the overall correlation relationship among distance, latency and energy consumption of the entire topology network, we derive the formula as shown in (11) and (12):

$$Corr = \sum_{i=1}^{na} \frac{\operatorname{cov}\left(T_{latency}, Gd_i\right)}{\sqrt{V_{latency} + V_{Gd}}}, (i = 1, 2, \ldots, na), \tag{11}$$

$$E \propto Corr_{n\alpha}, \tag{12}$$

where $Corr$ is the distance-latency correlation, $T_{latency}$ is the communication latency, $Gd$ is the ground distance, $n\alpha$ is the length of data path, and $V_{latency}$ and $V_{Gd}$ are the latency variance and the distance variance, respectively. The correlation $Corr$ can be described by a classic correlation. We use $E$ to represent the energy consumption of the topological network.

## 4 | LC-STREAM SYSTEM AND SCHEDULING STRATEGY

This section introduces the algorithms that implement the Lc-Stream scheduling strategy while considering the communication and workload during system runtime.

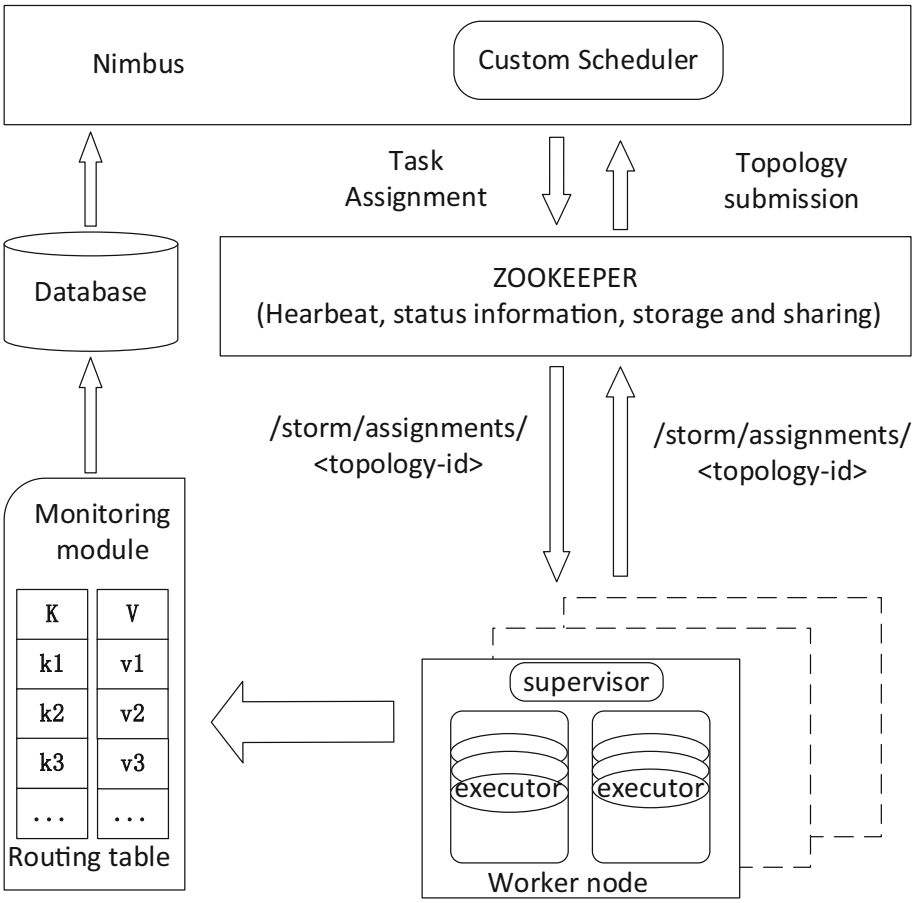


**FIGURE 2** Lc-Stream overview.

## 4.1 | Lc-Stream framework

Figure 2 presents an overview of the Lc-Stream system. Nimbus is mainly responsible for resource scheduling and task allocation. Supervisor receives Nimbus's task allocation and starts and stops worker processes (the number is set by the configuration file), worker runs a process that specifically handles component logic.

ZOOKEEPER provides a distributed coordination mechanism used to coordinate the work of all the nodes in a Storm cluster. ZOOKEEPER creates data when assigning tasks to the Topology for the first time. It stores the task information assigned by Nimbus to the Topology, including the local storage directory of the Topology on the Nimbus host and the assigned Supervisor machine to the hostname. To monitor each executor, ZOOKEEPER records which worker each executor is running on and the startup time of each executor. The worker node's data are then updated during the running process, and the worker will execute the tasks assigned to it according to the task assignment information saved in Database.

In addition, a monitoring module is introduced to monitor the arrival rate $\lambda$ of the data stream and the processing rate $\mu$ of tuples for the execution program and store them in the routing table. The storage structure is a map collection with the node number as the key and the rate as the value.

We deploy our Lc-Stream scheduler algorithm in the "Custom Scheduler" of the storm system.

## 4.2 | Resource allocation

Task is a basic logical execution unit in Storm.[4] When a topology finishes traversal and starts to run, the number of tasks cannot be changed. According to the Lc-Stream scheduling strategy, different physical machines have different IDs, and the executors hosted by the same physical machine provide the same physical machine ID. The maximum number of executors is set in the configuration file of each physical machine. These executors are numbered in the executor filter pool, stored in descending order, and allocated by the Lc-Stream scheduling strategy. In the context of a distributed streaming environment, tasks of the same type occupy the executors of the same physical machine. CPU resources are distributed among the worker nodes on different physical machines. Redundant tasks are redistributed according to the actual load status (the specific load status is introduced in Section 3). For the executor of each worker node, a "use" status bit is added as a mark. This marking method divides the executors according to their usage of resources mainly based on the number of occupied cpu. For example, the unavailable is 1, the available is 0. As we mentioned in Section 3.2,

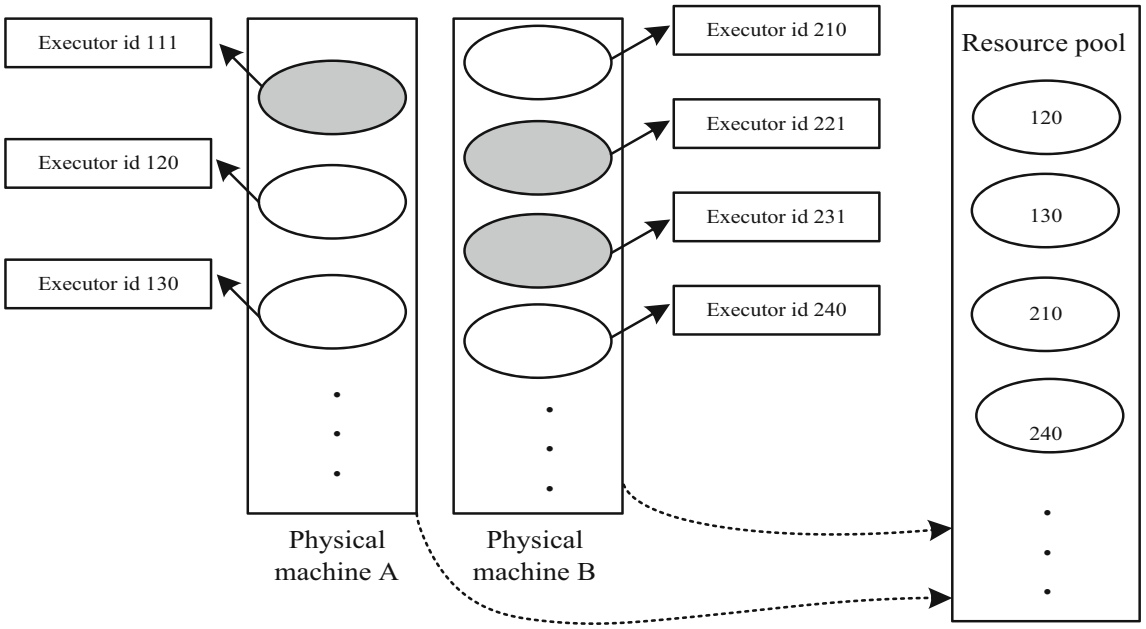


**FIGURE 3** Resource labeling schematic.

use the monitoring module to monitor resource occupancy. The system resource threshold is set. An overloaded executor does not participate in the sorting in Figure 3.

Figure 3 is an example. In machine A, there are three executors with id 111, id 120 and id 130. There are four executors in machine B with id 210, id 221, id 231 and id 240. The third digit of id is the status signal bit. As can be seen, executors with id 111, id 221 and id 231 are not available. The remaining ids are available executors. Reclaim these available executors into the resource pool.

To recycle the resource pool of the executors used, its status is changed from 1 to 0, and the flag becomes available. The host ID and executor keep unchanged. When picked from the resource pool, the reused executor is allocated according to the specific conditions of the system, and the allocation is only based on the CPU resources consumed by each physical machine in the system. Algorithm 1 describes the process:

**Algorithm 1.** Resource allocation algorithm

**Input:** operator set $N_i\{i = 1, 2, \ldots, k\}$, *DAG*.
**Output:** operator sorted set

**or** each $N_i$ in *DAG* **do**
    $n \leftarrow$ the number of executors
**end for**
*/*Assign an id to each executor. Define the serial number of the server. Sorted by resource allocation in descending order.*/*
**for** each $E_{id}$ **do**
    $E_{id} \leftarrow$ concatenate $p_{id}$, $e_{id}$ and $s_{id}$
    $s_{id} = 0$ or 1
    $\{\} \leftarrow$ sorted set of executors
**end for**
*/*Check whether the executor is available*/*
**if** $e_{id} < e_{\max}$ || $s_{id} = 0$ **then**
    $E_{id} \leftarrow$ choose the right node
    **while** $p_{id} == physical_{id}$ **do**
        $\rho = \lambda/\mu$
        $p_0 = \rho$
**end while**
**end if**
**return** sorted set of executors{}

The input of the algorithm includes the application DAG (Directed Acyclic Graph) and the operator set $N_i\{i = 1, 2, \ldots, k\}$, and the output is the logical solution after resource allocation. We first assign an id to each executor and define the serial number of the server in step 6. Based on the current resource allocation of the server where the executors are located, the executors on each physical machine are sorted in descending order in step 8. The id is recorded in the resource pool in sequence, a state bit is added to each executor, and the state is set as available. Finally, according to the availability of executors in the executor resource pool and the degree of resource occupation, tasks are selected for allocation from step 11 to 17.

By now, the tasks can consume reasonable and sufficient resources to execute the DAG in a low-latency and stable way. Although the time complexity of this resource allocation algorithm is O(n) which is quite optimal, it does not consider the impact of subsequent component communication problems, so this algorithm needs to be further improved.

## 4.3 | Node selection

A monitoring module is added to the Lc-Stream scheduling strategy to detect the arrival rate of tuples and model the executor processing time and the available resources in the system. According to the arrival rate and processing rate, a state-related queuing model is adopted.

As sum that the data arrival rate is $\lambda$, the processing rate of the executor is $\mu$, the resource consumption of a single executor is $c_k$, and the arrival rate and processing rate of the actual stream computing system fluctuate in real-time. The arrival rate and processing rate of the actual stream computing system change in real time. For example, the arrival rate is too high, the Spout component is too stressed, and the distribution speed is slow. Sometimes the system needs to block pulling data from the data source because CPU resources are occupied. These situations are addressed by improving the model.

The expected value $E_n$ can be described by (13).

$$E_n = c_k\mu + W_k L_k = c_k\mu + W_k \frac{\lambda}{\mu - \lambda}, \tag{13}$$

where $W_k$ said the average waiting time for data tuples, $L_k$ is expressed as the average queue length of data tuples. The worker node optimal processing rate $\mu^*$ can be calculated by (14).

$$\mu^* = \lambda + \sqrt{\frac{W}{c_k}\lambda}. \tag{14}$$

The entire cluster $E$ can be calculated by (15).

$$E = c'_k k + W_k L. \tag{15}$$

Use the marginal analysis method to get (16).

$$\begin{cases} c'_k k^* + W_k L(k^*) \le c'_k(k^* - 1) + W_k L(k^* - 1). \\ c'_k k^* + W_k L(k^*) \le c'_k(k^* + 1) + W_k L(k^* + 1) \end{cases} \tag{16}$$

Obtain results can be calculated by (17).

$$\Delta L = L(k^*) - L(k^* + 1). \tag{17}$$

when $\Delta L$ is the minimum obtained by satisfying the above formula, $k^*$ is the number of managed processors at the optimal processing rate.

The queuing models determines the operating status of each executor, monitors the entire system, and records the load criticality caused by the high arrival rate $\lambda$ and the low processing rate $\mu$. Lc-Stream's node selection algorithm optimizes and it records the system status. If the same situation appears, corresponding scheduling can be applied directly. Algorithm 2 describes the process of node selection.

**Algorithm 2.** Node selection

**Input:** arrival rate $\lambda$, processing rate $\mu$

**Output:** number of managed processors $k^*$

1. /* *Calculate the ideal processing rate for each executor*/*
2. **For** each $e_i$ executor **do**
3. $\mu^* = \lambda + \sqrt{\frac{W}{c_k}\lambda}$
4. $\mu \leftarrow \mu^*$
5. **end for**
6. /* *Solve for $\Delta L$ by (14) and (15). The minimum value of $\Delta L$ is used to obtain the optimal number of executor.*/*
7. **for** all k in 1, 2, … , $N$ **do**
8. if $c'_k k^* + W_k L(k^*) \le c'_k(k^* - 1) + W_k L(k^* - 1)$ then
9. if $c'_k k^* + W_k L(k^*) \le c'_k(k^* + 1) + W_k L(k^* + 1)$ then
10. $\Delta L = L(k^*) - L(k^* + 1)$;
11. end if
12. $\Delta L' = \min(\Delta L', \Delta L)$
13. end if
14. $k \leftarrow$ number of processors
15. **end for**
16. $k^* = k$
17. **return** optimal number of processors $k^*$

The input of the algorithm includes data arrival rate $\lambda$ and executor processing rate $\mu$, and the output is an optimal number of processors $k^*$. According to the arrival rate $\lambda$, the algorithm calculates the ideal processing rate $\mu^*$ of the node by substituting it into Equation (14), which are steps 1 to 5 in Algorithm 2. Marginal analysis is used in Equation (16) to determine the quantified range of resource consumption for a single task (from steps 8 to 13). Difference $\Delta L$ measure as a node selection index. When $\Delta L$ value is smaller, suggests that at this point in the cluster resource allocation more reasonable (from steps 6 to 16). Record the most reasonable number of executors $k^*$ and $k^*$ in this state is obtained, i.e., the optimal degree of parallelism.

## 4.4 | Low-latency communication based on component interaction

The number of executors in the system is enabled according to the default number of Storm. Our subsequent allocation method is based on the actual usage of CPU resources in order to achieve the system load balance. In the scheduling process, it may be found that the associated tasks in the upstream and downstream components in the logical structure are not allocated to the same physical machine slot or even the same physical machine. The cost of inter-communication is too high.[41]

As shown in Figure 4, machine A and machine B are two worker nodes connected through the network. The executor $a_1$ and the executor $a_2$ are two different executor processes, executing on the same node (machine A). Within the device $a_1$ is the communication between threads. Figure 4 shows the communication within the cluster, including the communication between threads, between executor processes, and between remote worker nodes.

The transmission cost between different physical machines is the highest, followed by the cost between nodes, then the cost within the same process which is the smallest. The relevant constraints are detailed below:

$$E = \{e_i\}, \mathrm{i} = 1, 2..j.., N, \tag{18}$$

$$W = \{w_i\}, \mathrm{i} = 1, 2..j.., N, \tag{19}$$

$$\forall_{i,j} \in N, \underset{(e_i,e_j \in E)}{\theta_{e_i,e_j}} < \underset{(w_i,w_j \in W)}{\theta_{w_i,w_j}}, \tag{20}$$

$$\forall_i \in N, L_{B(w_i)} < S, \tag{21}$$

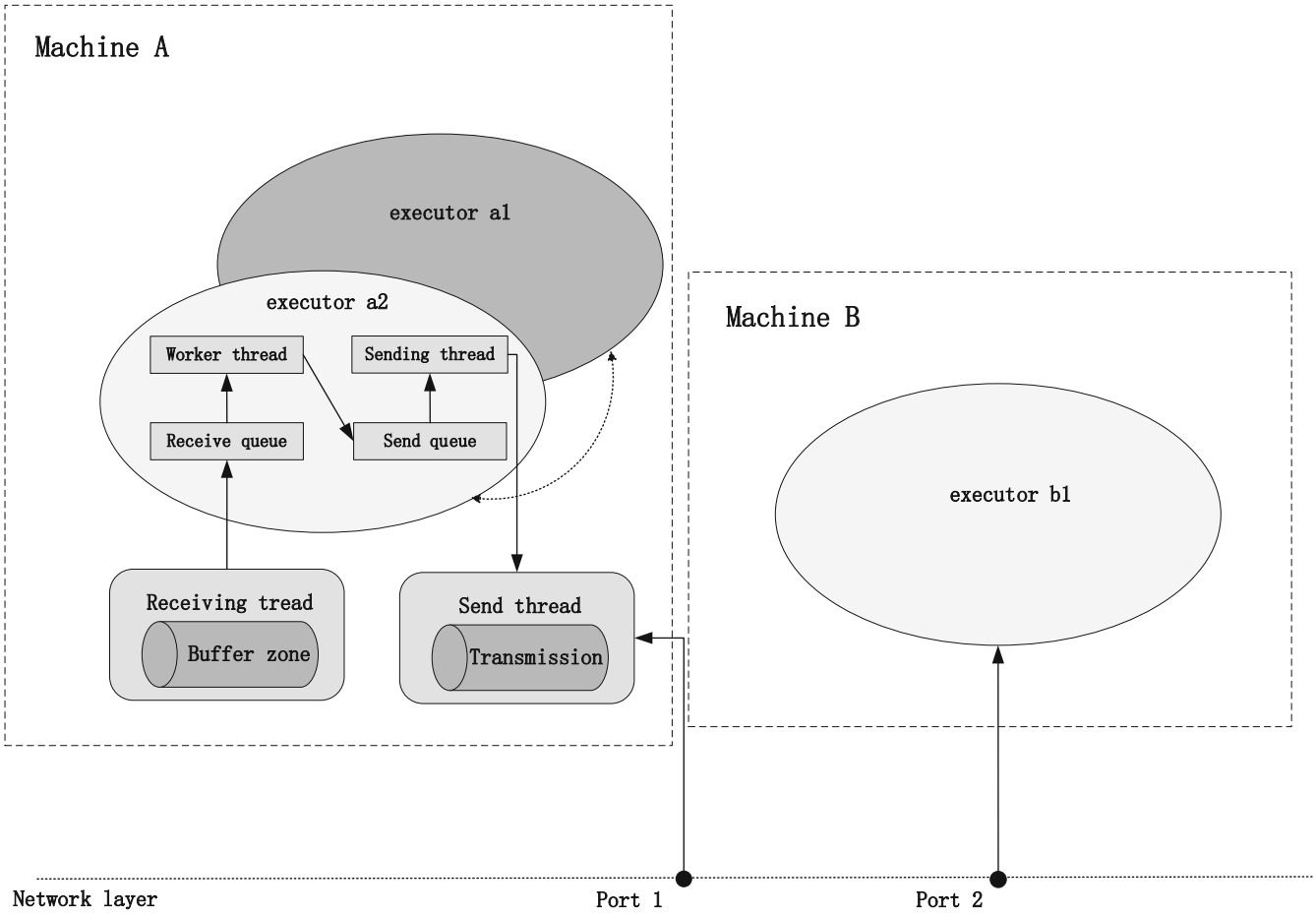


**FIGURE 4** Data transfer between executors on different physical machines.

where $N$ represents the total number of master node and zookeeper, $E$ and $W$ are the set of executors and workers respectively. $\theta$ represents the communication cost between worker nodes, $L_B$ represents the load of a single node and $S$ is the maximum load of worker nodes.

Lc-Stream tries to put related tasks in the same process as much as possible to load balance of the CPU resources. Each physical machine has relatively more resources to complete the above operations. When there is nothing to allocate to the same process, the resources are placed under the same physical machine according to the host ID to minimize the communication latency caused by inter-machine transmission. Algorithm 3 describes executor allocation algorithm.

**Algorithm 3.** Executor allocation algorithm

**Input**: the average waiting time $W_k$, the average queue length $L_k$

**Output**: the number of executors per node after rebalancing

1. *spn* ← *split number*
2. *pbolt* ← *partition bolt number*
3. *cbolt* ← *count bolt number*
4. */* based on the current latency*/*
5. $N_e = spn + pbolt + cbolt$
6. $L_k = \sum_{n=0}^{\infty} np_n = \sum_{n=0}^{\infty} \frac{n\rho_n}{n!} p_0 = \rho$
7. $W_k = \frac{L_k}{\lambda_e} = \frac{\rho}{\mu(1-e^{-\rho})}$
8. */* If the while condition is met, the redistribution will be carried out. */*
9. **While** $L_k > L$ && $W_k > W$ **do**
10. **If** $(L_k > L$ && $k < N_e)$ **then**
11. $k++$

12. **else**
13. $k = N_e$
14. **end if**
15. $a = k$
16. **if** ($W_k > W$ && $k < N_e$) **then**
17. $k++$
18. **else**
19 $k = N_e$
20. **end if**
21. $b = k$
22. **end while**
23. /* *Take the minimum values of a and b to lower the adequately resourced component executor. Before this, when the data volume reached the maximum load, the value of a or b was increased.*/
24. $k = \min(a, b)$
25. **return** rebalance executors number $k$

Algorithm 3 reallocates executors for components with excessive pressure. When a component's average wait time and average queue length are too long, the system will detect the problem and make adjustments. The input includes the average waiting time $W_k$, the average queue length $L_k$, and the executor allocation of the current component. First, the number of executors for each component is recorded. Where, spn saves the number of executors for distributing components, pbolt saves the number of executors for dividing components, and cbolt saves the number of executors for counting components (from step 1 to 3). Then, calculate the average queue length and average waiting time for each component (from step 6 to 7). When $W_k$ and $L_k$ are greater than the critical value, the redistribution will be carried out (step 9). Where, $a$ and $b$ record the number of executors respectively. In this case, the number of executors of the components with sufficient resources will be downregulated and added to the components with high pressure (from step 8 to 21). This will reach a new equilibrium. When the data traffic continues to increase and reaches the maximum load limit, the system selects the maximum number of executors that can be used.

## 4.5 | Energy saving based on geo-distributed stream computing environment

Through the analysis of energy saving model, we derive an approximate relationship between distance delay and network delay estimation methods. Lc-Stream's energy consumption optimization strategy involves selecting an appropriate server scale for the cluster while ensuring system efficiency. The trigger time is determined based on the proportion of server CPU utilization below a certain threshold to all cluster servers, which adjusts the number of cluster servers. Each server in the cluster starts a single thread to monitor CPU utilization and report it back to the master node promptly. The execution process is depicted in Figure 5.

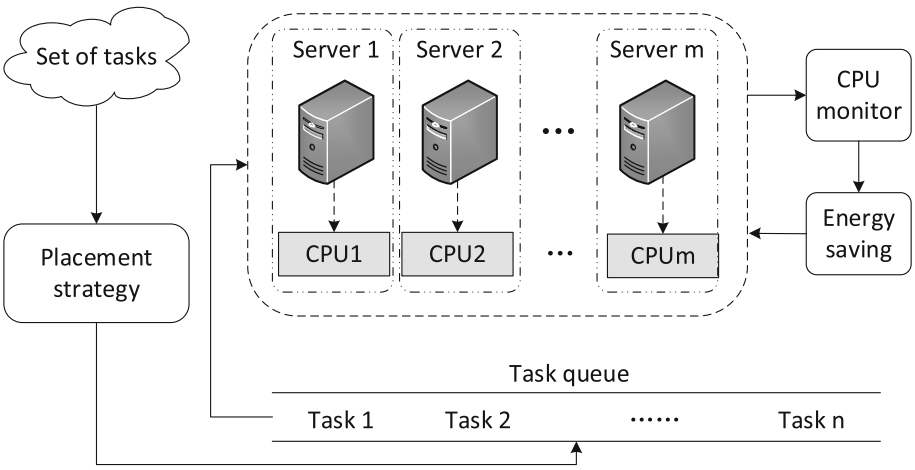


**FIGURE 5** Task distribution, CPU monitoring, and energy saving processes.

Figure 5 depicts the internal process and monitoring situation of a system server cluster after receiving distribution tasks. When the servers are monitored to have low CPU utilization, the energy-saving strategy will be triggered. We base the process of putting some servers into sleep mode on their task correlation and latency situation. When the number of servers with low CPU utilization under the quota accounts for a proportion of the total cluster server ratio that is higher than the default value, the server sleep-out will be stopped. We found through multiple testing experiments that when the server CPU utilization is below 0.25, the worker nodes will have fewer tasks to place and process. Placing these tasks on other worker nodes will not cause significant pressure to the system. Meanwhile, we specify that Lc-stream performs energy consumption optimization when the proportion of servers with CPU utilization below 0.25 in the system cluster reaches 0.7 of the entire server cluster.

Algorithm 4 describes energy saving algorithm.

**Algorithm 4.** Energy saving algorithm

**Input**: Cluster server CPU utilization set {}
**Output**: Dormancy server set {}

1. $N \leftarrow$ *the total number of executors*
2. $n \leftarrow$ *the number of low utilization executors*
3. **for** *each* $c_i$ *in* $N$ **do**
4. **if** $c_i < 0.25$ **then**
5. $n++$
6. **end if**
7. **end for**
8. $\eta = \frac{n}{N}$
9. **if** $\eta < 0.7$ **then**
10. *continue*
11. **else**
12. **while** $L_k > L$ && $W_k > W$ **do**
13. $\{\} \leftarrow$ *hibernate server*
14. **end while**
15. **end if**
16. **return** *hibernate server* {}

Algorithm 4 outlines the server size adjustment process in the cluster. It takes the CPU utilization of each server monitored by the monitoring module, as input and outputs the set of dormant servers. First, it calculates the number of executors working on the cluster server and identifies the executors with CPU utilization below 0.25. When the ratio of such executors to the total work executors exceeds 0.7, the optimization strategy is triggered. This ensures that the average waiting time and queue length of cluster executors remain within a reasonable range, primarily considering system latency due to associated task allocation on the same server. Finally, it outputs the set of dormant servers.

## 5 | PERFORMANCE EVALUATION

The proposed Lc-Stream system is implemented on top of Storm 2.1.0 in Ubuntu 20.04.1. Real-world data experiments are conducted in the Alibaba Cloud Computing cluster. The cluster consists of 30 machines, with 1 designated machine serving as the master node, running Storm Nimbus, 2 designated as Zookeeper nodes, and the rest 27 machines as Supervisor nodes. The software configuration of Lc-Stream platform is shown in Table 3.

We submitted two topologies to the system, WordCount and Top-N, and all subsequent experiments showed both topologies running simultaneously in the storm cluster. In order to obtain more accurate results, our experiment was recorded from the stable operation of the system.

WordCount is an application counting the frequency of words in English text. It consists of a Spout and two Bolt. It is typical stream application and used as the experimental topology to meet the heterogeneity requirement. The logic graph of WordCount is shown in Figure 6. Spout reads data from an external data source ($V_{a1}$) and sends a random tuple object out (s). Bolt receives the tuple object output from Spout, splits the data in the tuple into words ($V_{b1}$), and emits the segmented words (f). Bolt receives the word array output from the previous bolt, accumulates the frequencies of the words in it ($V_{c1}$), and outputs the accumulated result.

One Top-N DAG is submitted at the same time to the computing cluster. The logic graph of Top-N is shown in Figure 7.

According to the distribution of existing servers, we construct the topological diagram of the network, as sown in Figure 8.

We conducted comparison between Lc-Stream and R-Storm[27,42] and Ts-Stream,[32] as these works are recent and closely related to our research.

**TABLE 3** Software configuration of Lc-Stream.

| Software | Version |
|---|---|
| OS | Ubuntu 20.04.1 |
| Storm | Apache-storm-2.1.0 |
| JDK | Jdk1.8 64bit |
| Zookeeper | Zookeeper-3.4.14 |
| Maven | Maven-3.6.2 |
| MySQL | MySQL-8.0.19 |
| Network | 1Gbps |

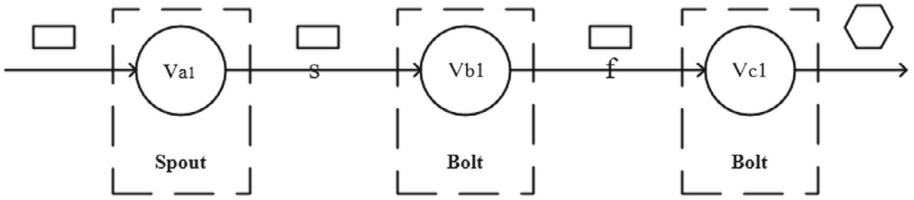


**FIGURE 6** Logical graph of WordCount.

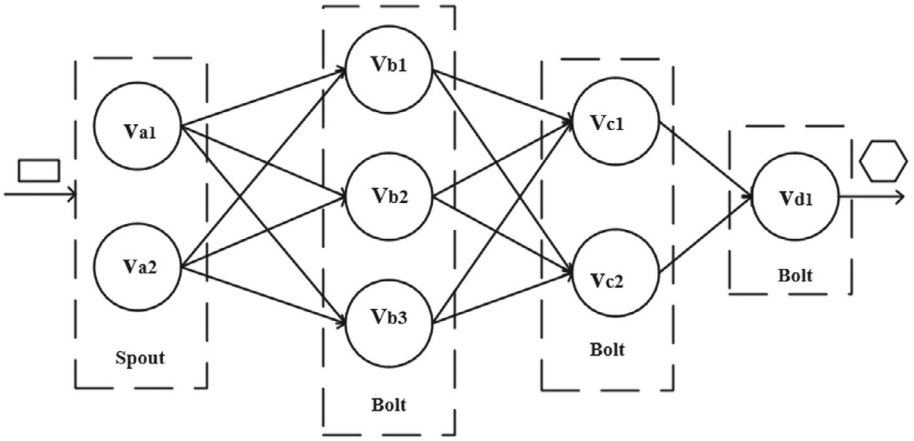


**FIGURE 7** Logical graph of Top-N.

## 5.1 | Process latency

As shown in Figures 9 and 10, we evaluate the main logical components (spouts or bolts) in the topology. We observe the latency for the 'count' operator and 'split' operator to examine the impact of Lc-Stream scheduling on the latency of performers.

In Figure 9, the system starts with low latency. As time elapses, after the system runs for approximately 2 min, Ts-Stream exhibits a significant reduction in latency. In contrast, R-Storm maintains relatively high latency, while Lc-Stream's latency starts to decrease. Once the system stabilizes, Lc-Stream's latency is approximately 21.5% lower than that of R-Storm and about 5% lower than that of Ts-Stream. All the three systems maintain low latency operations. By the 7th minute, the latency fluctuations of the three are minimal, with average values of approximately 5.8 ms, 5.35 ms and 5.1 ms, respectively. After that, Lc-Stream continues to reduce latency further. In summary, when compared to R-Storm and Ts-Stream, Lc-Stream can reduce the total system latency by 20% to 30% and 4% to 10%, while keeping nearly the same capacity.

In Figure 10, latency for the "split" operator is the lowest under Lc-Stream when compared to the other two systems. Within the first minute of topology execution, as Lc-Stream functions to buffer data, the average latency is reduced by approximately 19.1% when compared to R-Storm and 5.9% when compared to Ts-Stream. It is conducive to the overall stability of the system. In the subsequent operation process, Lc-Stream has slight

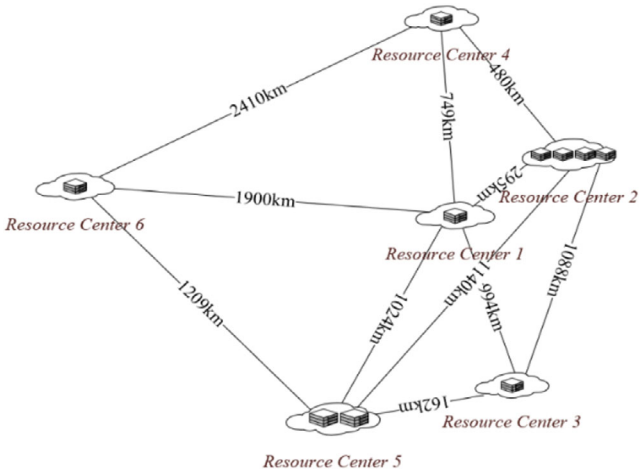


**FIGURE 8** Network topology diagram.

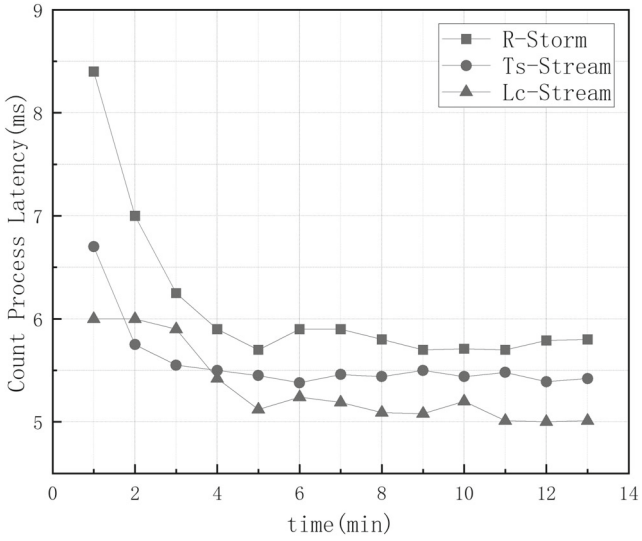


**FIGURE 9** "Count" process latency of R-Storm, Ts-Stream and Lc-stream.

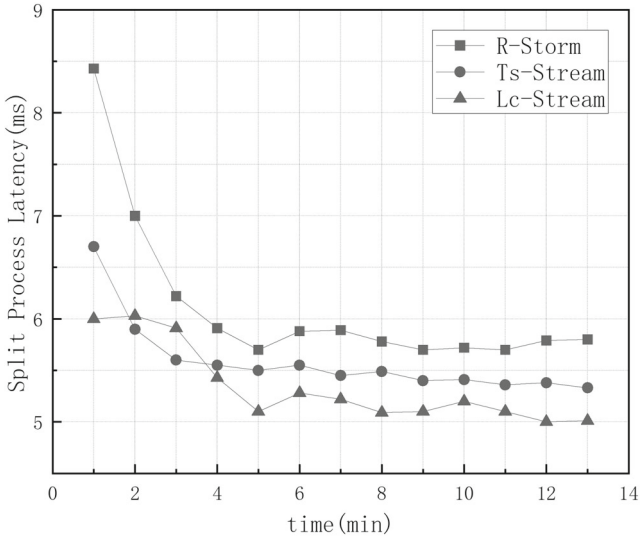


**FIGURE 10** "Split" process latency of R-Storm, Ts-Stream and Lc-stream.

latency fluctuations. The reason might be the change to the average arrival rate and processing rate of the data stream causes the expected value and the parallelism changes. It is stabilized in a short time. No large latency fluctuation is caused.

## 5.2 | System latency

We compare the system execution latency, processing latency and total system latency. The results are shown in Figures 11–13.

Figure 11 displays the system execution latency results under R-Storm, Ts-Stream and Lc-Stream. Execution latency describes the overall execution latency of a submitted topology. It includes the latency time of tuple distribution to the executor and the latency produced by the communication of components in the scheduling process. The overall data stream stays relatively stable without big fluctuation. As we can see, both of them keep a relatively stable state. Compared to R-Storm and Ts-Stream, Lc-Stream scheduler reduces the system execution latency by approximately 35% and 19%, respectively.

System processing latency describes the overall processing latency of a submitted topology in the system. It mainly includes the time consumed by each executor for task processing. In terms of system processing latency (Figure 12), the Lc-stream scheduler is 28.5% and 15.5% lower than R-Storm and Ts-Stream, respectively.

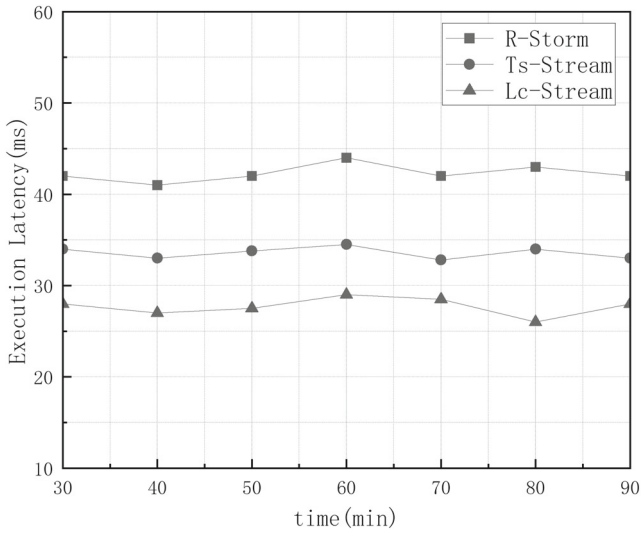


**FIGURE 11** System execution latency of R-Storm, Ts-Stream and Lc-Stream.

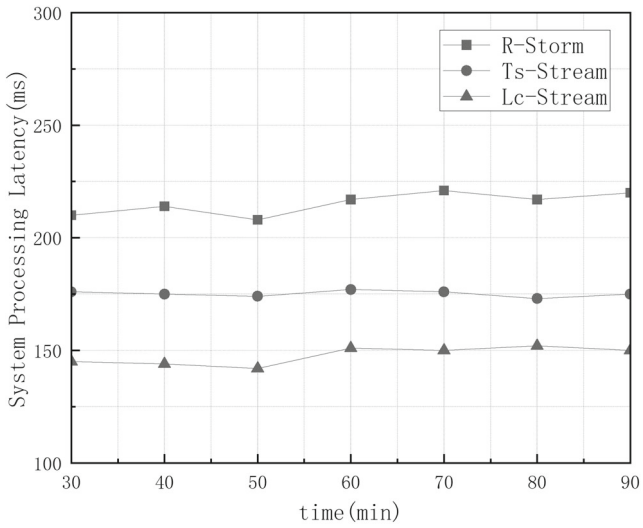


**FIGURE 12** System processing latency of R-Storm, Ts-Stream and Lc-Stream.

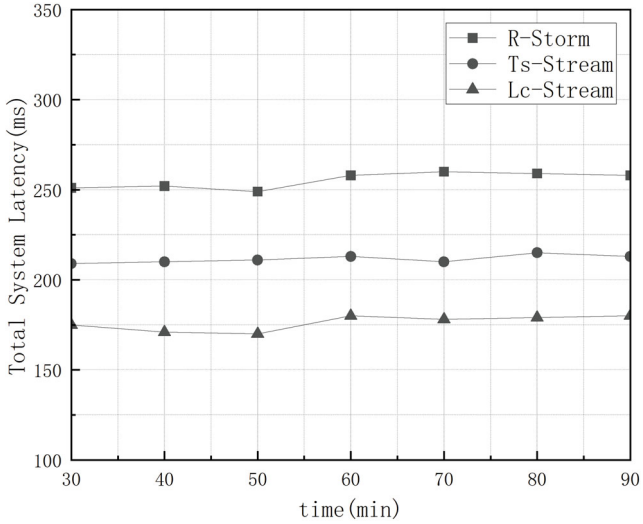


**FIGURE 13** Total system latency of R-Storm, Ts-Stream and Lc-Stream.

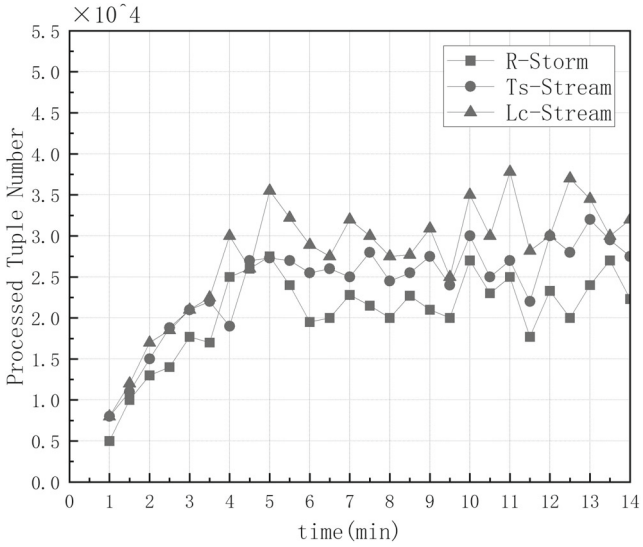


**FIGURE 14** Processed tuple number by R-Storm, Ts-Stream and Lc-Stream.

Total latency of a system is roughly the sum of system execution latency and system processing latency. In Figure 13, the Lc-stream scheduler reduces the total system latency by 30.7% and 16.7% compared to R-Storm and Ts-Stream, respectively. From the above figures, it can be seen clearly that Lc-stream scheduler outperforms R-Storm and Ts-Stream in terms of system latency.

## 5.3 | Throughput

We take the number of tuples processed within a given period of time as an indicator to measure the system throughput. In Figure 14, the x axis is the time range (14 min) and the y axis represents the number of tuple processed. In the beginning, the throughput of Lc-Stream is slightly higher than that of R-Storm and Ts-Stream, but the difference is not significant. After approximately 5 min, Lc-Stream consistently processes a higher number of tuples compared to R-Storm and Ts-Stream, establishing the highest the throughput among the three.

The three strategies use the same number of worker nodes in the comparative experiment. Based on the analysis of the data collected by the monitoring module, a more reasonable allocation scheme is selected. The above experiments show that our scheduling scheme can indeed

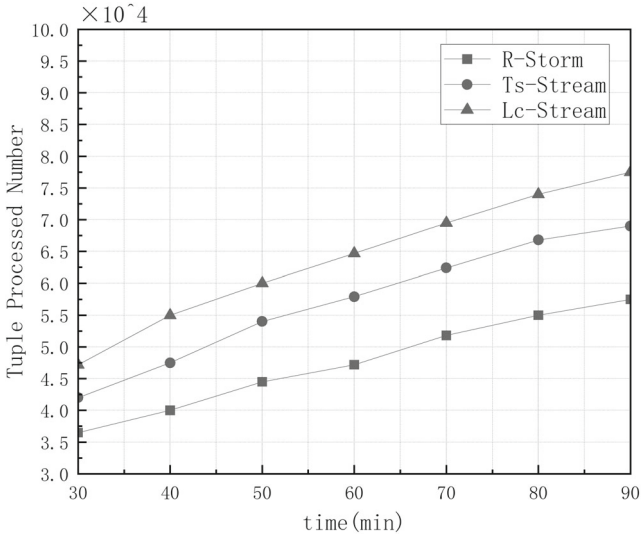


**FIGURE 15** System throughput of R-Storm, Ts-Stream and Lc-Stream.

improve the throughput of the system. Compared to R-Storm and Ts-Stream, the throughput of Lc-Stream system is increased by 47% and 21.7%, respectively.

Similarly, we also conduct experiments on system throughput in a Geo-Distributed environment for R-Storm, Ts-Stream and Lc-Stream. The throughput experiment is in the same environment as the system latency experiment. We recorded the changes in system throughput along with the various latency.

As shown in Figure 15, the experimental data for throughput comparisons between R-Storm, Ts-Stream and Lc-Stream systems are recorded. For the measurement of throughput, the experiment is calculated according to the number of tuples emitting, forwarding, executing and failing in the tuples of the upstream and downstream components of the system. We refer to the number of successfully emitted tuples as throughput. We recorded the data during the experiment through Storm UI. The refresh time is 1 minute. We keep track of the number of tuples per 10 minutes. Lc-Stream consumes 34.8% and 15.8% more tuples per minute than R-Storm and Ts-Stream, respectively. Experiments show that Lc-Stream can improve system throughput in Geo-Distributed environment.

## 5.4 | System energy consumption

In this part, we conducted relevant experiments on the energy consumption of Lc-Stream and R-Storm in Geo-Distributed clusters. We use the POWERTOP tool to monitor the power consumption of each of the 27 servers. Additionally, we monitor the CPU utilization of each server individually for clarity. The performance of R-Storm and Lc-Stream scheduling strategies is compared under identical topology and server conditions. The Lc-Stream scheduling strategy changes component (spout or bolt) parallelism by adjusting the number of executors for each.

The CPU utilization of the 27 servers is shown in Figure 16.

Figure 16 shows the CPU utilization during the experiment for both R-Storm and Lc-Stream. Generally, higher CPU utilization corresponds to higher power consumption and energy consumption. R-Storm maintains relatively low average CPU utilization, with the highest average CPU utilization not exceeding 26%, resulting in a surplus of idle resources and unnecessary energy consumption. In contrast, Lc-Stream maintains CPU utilization between 30% and 40%. Lc-Stream defines an energy consumption threshold, triggering a rebalancing mechanism when CPU utilization is low or system latency is too high. A new allocation policy is then calculated. Starting from the 16th server, CPU utilization tends towards 0. Idle servers are put to sleep according to the energy saving algorithm. A similar trend is observed in the power consumption (Figure 17) based on our monitoring data.

As shown in Figure 17, the power consumption of Lc-Stream among the first 15 servers is relatively high. This is a result of increased CPU power consumption. But for the entire system, 12 servers are put to sleep during the operation of Lc-Stream to minimize energy consumption. Lc-Stream reduces the total energy consumption of the system by 17.7%. Sleeping idle servers can effectively reduce energy consumption.

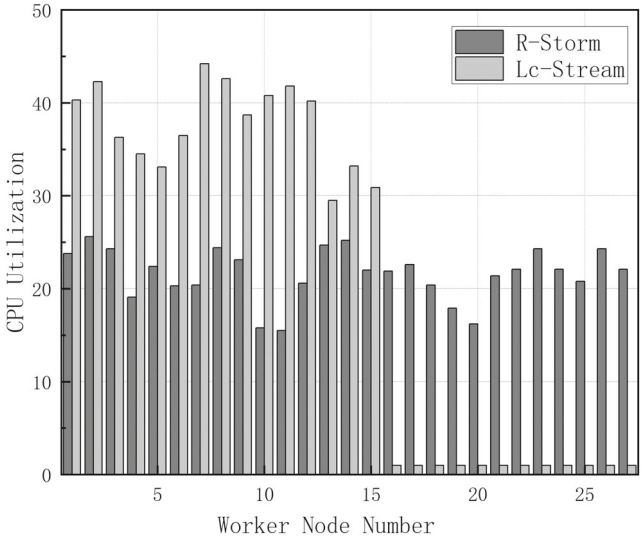


**FIGURE 16** CPU utilization of R-Storm and Lc-Stream.

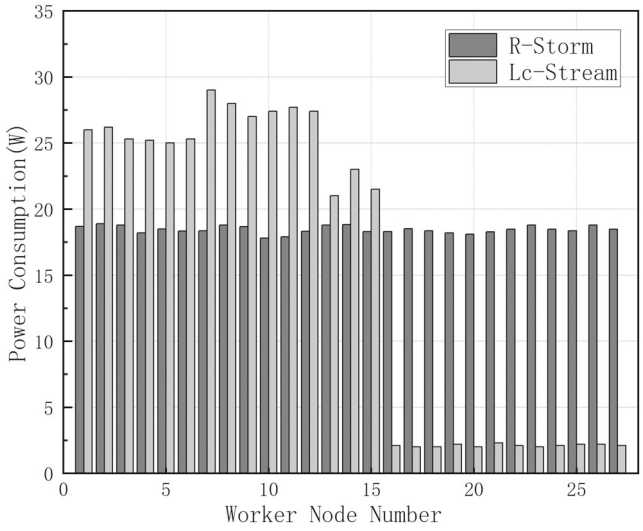


**FIGURE 17** Power consumption of R-Storm and Lc-Stream.

## 6 | CONCLUSIONS AND FUTURE WORK

Low latency and high throughput are two of the most critical performance requirements for big data stream computing systems, especially in Geo-Distributed environments. It is of vital importance to investigate low latency scheduling strategy. In this article, we propose an elastic scheduling strategy with Latency Constraints in Geo-Distributed stream computing environments called Lc-Stream. Our work can be summarized as follows: (a) We use queued network theory to optimize the data stream redirection method to construct the model. There are computing resource model, latency constrained scheduling model and communication energy consumption model. (b) A modified node selection method based on the relevance of the task, to reduce the communication latency between groups at the executor granularity. (c) a network cluster distribution constructed in a Geo-Distributed streaming computing environment, to measure energy saving schemes and ensure low transmission latency.

The proposed Lc-Stream and the relevant scheduling strategies are implemented and evaluated on a real-world distributed stream computing platform. Due to its reliable and powerful real-time processing capabilities, Storm is widely used by major platforms. We configured the Apache Storm system for the Geo-Distributed environment to conduct the experiment. In the same experimental setting, we compare the Lc-Stream strategy with two other strategies, R-Storm and topology-aware scheduling, to assess their performance in terms of latency and throughput. The experimental results show that compared with the other two methods, Lc-Stream reduces the total latency by more than 19% (compared to R-Storm)

and 15% (compared to Ts-Stream) on a typical Geo-Distributed multi-task topology, and increases throughput by more than 37% (compared to R-Storm) and 21% (compared to Ts-Stream). It is proved that using our Lc-Stream scheduling framework can significantly improve the low latency and high throughput performance of Geo-Distributed stream systems.

Our future work will be focusing on the following.

1. We will consider the processing efficiency in the case of Geo-Distributed heterogeneous clusters, attempting to improve the efficiency and distribution methods of complex class clusters, and
2. Applying Lc-Stream in a real application environment, such as online shopping, online social platforms, online search and other big data streaming computing scenarios, and testing its latency and throughput to verify its feasibility and effectiveness for potential improvement.

## ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under Grant No. 62372419; the Fundamental Research Funds for the Central Universities under Grant No. 265QZ2021001; and Australian Research Council (ARC) Discovery Project.

## CONFLICT OF INTEREST STATEMENT

The authors declare no conflicts of interest.

## DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

## ORCID

*Dawei Sun* https://orcid.org/0000-0003-3137-6257
*Rajkumar Buyya* https://orcid.org/0000-0001-9754-6496

## REFERENCES

1. Liu S, Weng J, Wang JH. An adaptive online scheme for scheduling and resource enforcement in storm. *IEEE/ACM Trans Netw*. 2019;27(4):1373-1386.
2. Hidalgo N, Wladdimiro D, Rosas E. Self-adaptive processing graph with operator fission for elastic stream processing. *J Syst Softw*. 2017;127(C):205-216.
3. Liu G, Jiang C, Zhou M. Time-soundness of time petri nets modelling time-critical systems. *ACM Trans Cyber-Phys Syst*. 2018;2(2):1-27.
4. Apache Storm. Accessed February 23, 2023. https://storm.apache.org/
5. Cardellini V, Grassi V, Presti FL, Nardelli M. Optimal operator replication and placement for distributed stream processing systems. *ACM Sigmetrics Performance Evaluat Rev*. 2017;44(4):11-22.
6. Liu X, Buyya R. Resource management and scheduling in distributed stream processing systems: A taxonomy, review, and future directions. *ACM Comput Surv (CSUR)*. 2020;53(3):1-41.
7. Zhou B, Li J, Xiaoyan Wang YG. Online internet traffic monitoring system using spark streaming. *Big Data Mining Anal*. 2018;1(1):47-56.
8. Sheng X, Tang J, Xiao X, Xue G. Sensing as a service: challenges, solutions and future directions. *IEEE Sens J*. 2013;3(10):3733-3741.
9. Tien JM. Internet of things, real-time decision making, and artificial intelligence. *Ann Data Sci*. 2017;4(2):149-178.
10. Di W, Ke Y, Yu JX. Detecting leaders from correlated time series. *Lect Notes Comput Sci*. 2010;5981(7):352-367.
11. Hopfgartner F, Kille B, Heintz T. Real-time recommendation of streamed data. Proceedings of the 9th ACM Conference on Recommendation Systems, Austria, Association for Computing Machinery, 361-362. 2015.
12. Lobato AG, Pastana AL, Martin CA. A fast and accurate threat detection and prevention architecture using stream processing. *Concurr Comput Pract Exp*. 2021;34(3):1-17.
13. Bugra G, Scott S, Martin H. Elastic scaling for data stream processing. *IEEE Trans Parallel Distrib Syst*. 2014;25(6):1447-1463.
14. Kalim F, Cooper T, Huijun W. Caladrius: A performance modelling service for distributed stream processing systems. 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macau SAR, China, 1886-1897. 2019.
15. Wei X, Li L, Li X. Pec: proactive elastic collaborative resource scheduling in data stream processing. *IEEE Trans Parallel Distrib Syst*. 2019;30(7):1628-1642.
16. Fu M, Agrawal A, Floratou A, et al. Twitter heron: towards extensible streaming engines. 2017 IEEE 33rd International Conference on Data Engineering (ICDE). IEEE, 1165-1172. 2017.
17. Tuor T, Wang S, Leung KK, et al. Online collection and forecasting of resource utilization in large-scale distributed systems. 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE: 133-143. 2019.
18. Li T, Zhiyuan X, Tang J, Wang Y. Model-free control for distributed stream data processing using deep reinforcement learning. *Distributed Parallel Cluster Comput*. 2018;11(6):705-718.
19. Zhiming H, Li B, Luo J. Time- and cost- efficient task scheduling across geo-distributed data centers. *IEEE Trans Parallel Distrib Syst*. 2018;29(3):705-718.
20. Li C, Tang J, Luo Y. Load balance based job scheduling in geo-distributed clouds. *Wirel Pers Commun*. 2019;107(1):169-192.
21. Li C, Liu J, Li W, Luo Y. Adaptive priority-based data placement and multi-task scheduling in geo-distributed cloud systems. *Knowledge-Based Syst*. 2021;224:107050.
22. Li C, Liu J, Wang M, Luo Y. Fault-tolerant scheduling and data placement for scientific workflow processing in geo-distributed clouds. *J Syst Softw*. 2022;187(C):111227.
23. Russo GR. Self-Adaptive data stream processing in geo-distributed computing environments. DEBS'19: Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems, Darmstadt Germany. 2019.

24. Chen W, Paik I, Hung PCK. Transformation-based streaming workflow allocation on geo-distributed datacenters for streaming big data processing. *IEEE Trans Services Comput*. 2019;12(4):654-668.
25. Aniello L, Baldoni R, Querzoni L. Adaptive online scheduling in storm. Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems, Association for Computing Machinery, New York, 207-218. 2013.
26. Xu J, Zhenhua C, Jian T. T-storm: traffic-aware online scheduling in storm. 2014 IEEE 34TH International Conference on Distributed Computing Systems (ICDCS 2014), 535-544. 2014.
27. Peng B, Hosseini M, Hong Z, Farivar R. R-storm: resource-aware scheduling in storm. Proceedings of the 16th Annual Middleware Conference, 149-161. 2015.
28. Tantalaki N, Souravlas S, Roumeliotis M. A review on big data real-time stream processing and its scheduling techniques. *Int J Parallel Emergent Distrib Syst*. 2019;35(5):571-601.
29. Fu TZJ, Ding JB, Ma RTB. DRS: dynamic resource scheduling for real-time analytics over fast streams. 2015 IEEE 35th International Conference on Distributed Computing Systems, 2015:411-420. 2015.
30. Wang LI, Fu TZJ, Ma RTB, Winslett M. Elasticutor: rapid elasticity for realtime stateful stream processing. Proceedings of the ACM SIGMOD International Conference on Management of Data, 573-588. 2019.
31. Tantalaki N, Souravlas S, Roumeliotis M, Katsavounis S. Pipeline-based linear scheduling of big data streams in the cloud. *IEEE Access*. 2020;8:117182-117202.
32. Li B, Sun D, Chau VL. A topology-aware scheduling strategy for distributed stream computing system. 2021 12th EAI International Conference on Broadband Communications, Networks and Systems. Springer, Cham, 132-147. 2021.
33. Sun D, Cui Y, Minghui W, Gao S. An energy efficient and runtime-aware framework for distributed stream computing systems. *Future Gener Comput Syst*. 2022;136:252-269.
34. Leila E, Jason M, Zhiyi H, David E. I-scheduler: iterative scheduling for distributed stream processing systems. *Future Generat Comput Syst*. 2021;117:219-233.
35. Hadian H, Farrokh M, Sharifi M, Jafari A. An elastic and traffic-aware scheduler for distributed data stream processing in heterogeneous clusters. *J Supercomput*. 2023;79(1):461-498.
36. Mohammadreza F, Hamid H, Mohsen S, Ali J. SP-ant: an ant colony optimization based operator scheduler for high performance distributed stream processing on heterogeneous clusters. *Expert Syst Appl*. 2022:191:116322.
37. Zhang H, Sun D, Sajjanhar A, Buyya R. A data stream prediction strategy for elastic stream computing systems. *Broadband Commun Netw Syst*. 2022;413(1):148-162.
38. Salem F, Schintke F, Schütt T, Reinefeld A. Scheduling data streams for low latency and high throughput on a Cray XC40 using Libfabric. *Concurrency Comput Pract Exp*. 2020;32(20):1-14.
39. Al-Sinayyid A, Zhu M. Job scheduler for streaming applications in heterogeneous distributed processing systems. *J Supercomput*. 2020;76:9609-9628.
40. Ma K, Liu S, Lin Y, Yu Z, Ji K. Parallel grouping particle swarm optimization with stream processing paradigm. Proceedings of the 19th IEEE International Conference on High Performance Computing and Communications Workshops, IEEE, Thailand, 22-26. 2017.
41. Nasir MAU, De Francisci Morales G, García-Soriano D, Kourtellis N, Serafini M. The power of both choices: practical load balancing for distributed stream processing engines. 2015 IEEE 31st International Conference on Data Engineering, Seoul, Korea (South), 2015:137-148.
42. Resource Aware Scheduler [Online]. 2019. Accessed August 23, 2023. http://storm.apache.org/releases/2.0.0/Resource_Aware_Scheduler_overview.html


**How to cite this article:** Sun D, Wang Y, Sui J, Gao S, Rong J, Buyya R. Lc-Stream: An elastic scheduling strategy with latency constraints in geo-distributed stream computing environments. *Concurrency Computat Pract Exper*. 2024;36(14):e8085. doi: 10.1002/cpe.8085