A Task Dependency-Aware Scheduling Strategy for Cross-Domain Stream Computing Environments

Dawei Sun^{1,*}, Zhongyuan Zhao^{1}, Yueru Wang^{1}, Shang Gao^{2}, Rajkumar Buyya^{3}
^{1}School of Information Engineering, China University of Geosciences, Beijing,10083, China
^{2}School of Information Technology, Deakin University, Waurn Ponds, Victoria 3216, Australia
^{3}Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
E-mail:sundaweicn@cugb.edu.cn, 18532517385@163.com, wangyueru@email.cugb.edu.cn, shang.gao@deakin.edu.au, rbuyya@unimelb.edu.au
Corresponding author: Dawei Sun

Abstract. In cross-domain stream computing, assigning highly dependent tasks to different domains causes poor performance. Existing methods ignore cross-domain and focus on load balancing and resource allocation. To address this scheduling challenge, this paper proposes a task dependency-aware scheduling strategy named Td-Stream. This strategy is discussed in the following aspects: (1) Impact analysis: Analyzing the adverse impact of communication dependencies between tasks on system performance under traditional scheduling methods in cross-domain environments. (2) Model construction: Constructing models for stream topology, task dependency, resource cost.(3) Cross-domain task allocation: Introducing a cross-domain dependent task allocation method that incorporates a resource elasticity mechanism. Experimental results demonstrate significant improvements made by Td-Stream compared to existing state-of-the-art works.

Keywords: task dependency, scheduling strategy, stream computing, cross-domain environment, distributed systems

I. Introduction

The value of big data is realized through the collection, storage, accurate analysis, and deep mining of data within the context of a cross-domain environment [1]. Batch processing, as a traditional big data processing method, offers the advantages of handling multiple jobs simultaneously, but it falls short in meeting the needs of real-time big data analysis. Consequently, distributed stream processing systems (DSPS) such as Spark Streaming, S4 [2], Samza [3], R-Storm [4], Apache Storm [5], and Apache Flink [6], with their specialized data processing mechanisms and architectures, have driven the development of data processing technologies and expanded their application fields.

Currently, the optimization of big data stream computing systems mainly focuses on five core challenges: resource scheduling [7], data migration [8], data partitioning [9], load balancing [10], and system fault tolerance [11]. Among these, resource scheduling strategies are particularly important as they ensure the efficient and stable operation of the system, and are central to achieving real-time data processing responses.

In a distributed cluster system involving cross-domain environments, the delay caused by frequent I/O operations is a core concern, especially when processing large volumes of streaming tasks that rely on I/O operations. Previous studies often overlook the impact of inter-task communication dependencies on system latency. However, cross-domain communication between server groups significantly affects performance. Therefore, optimizing the latency of cross-domain stream computing clusters and developing more efficient scheduling strategies are essential to meet the increasing complex data processing requirements.

Scheduling strategies are widely embedded in various mainstream real-time distributed stream computing frameworks [12]. However, due to the general nature of these strategies, improving their adaptability to multi-stream computing application environments has become a pressing issue in the industry [13]. For instance, in Apache Storm, the scheduler distributes tasks evenly across worker processes in a round robin fashion, which effectively balances the load but fails to consider task dependencies. This oversight negatively impacts system performance, leaving room for optimization in communication overhead [14], resource allocation [15], and other areas. Therefore, latency optimization in cross-domain stream computing clusters should be a focal point. Heuristic algorithms [16], approximate algorithms [17] should be utilized to find near-optimal solutions, with an emphasis on developing more efficient and precise scheduling schemes to handle the increasing data processing demands [18].

To address this scheduling challenge, this paper proposes a task dependency-aware scheduling strategy named Td-Stream. This strategy is discussed through the following four aspects:

(1)Refined partitioning and dependency modeling: Construct stream topology, quantify task dependencies, transform applications into graphs for allocation, introduce elasticity for monitoring. Partition DAG with enhanced algorithms considering communication degrees.

(2)Cross-domain allocation and resource elasticity: Use greedy algorithm for cross-domain mapping and prioritize resource allocation. Have a mechanism to identify bottlenecks, adjust parallelism, optimize allocation and minimize latency.

The rest of this paper is organized as follows: Section II introduces related work. Section III defines the problem and constructs models. Section IV details the task dependency-aware scheduling policy and explains Td-Stream scheduling strategy. Section V presents the experiment and performance analysis. Finally, Section VI concludes the paper and discusses future work.

II. Related Work

Scheduling is the process of finding an optimal matching between tasks and computing resources. It involves minimizing communication dependencies between tasks, optimizing overall resource utilization, and ultimately maximizing cost-effectiveness, all while satisfying a set of relevant constraints.

R-Storm [19] is a built-in resource-aware scheduler for Apache Storm. It is designed to enhance overall processing capability by optimizing resource utilization and reducing internal network latency. This scheduling approach formulates the resource-aware scheduling problem for Storm applications as a quadratic multivariate three-dimensional knapsack problem. However, R-Storm has relatively high computational overhead in distributed stream processing scenarios.

SP-Ant [20] is a scheduling method designed to address the operator placement problem. It utilizes the ant colony algorithm to determine the optimal allocation of executors to compute nodes by balancing local optimization with a global optimization strategy. While SP-Ant is well-suited for heterogeneous computing clusters, it lacks a more in-depth exploration of resource elasticity.

ER-Storm [21] addresses the challenges of resource elasticity and scalable decision-making. It seeks to mitigate the communication overhead incurred by operator elastic expansion through replication and relocation at runtime. This method discretizes the input workload and models the relocation of operators across compute nodes during the expansion decision process. Additionally, it introduces the concept of model-free reinforcement learning to find the optimal solution.

Cross-domain distribution is a distributed architecture model with core objectives that include managing large-scale data processing, ensuring high availability, and reducing access latency and other computing requirements. The model overcomes the limitations of a single geographic location by deploying system components, data storage, computing power, and network resources across geographically distant regions or multiple data centers. This approach improves resource utilization efficiency, data security, and business continuity. Consequently, cross-domain distribution has triggered significant research interest.

Chunlin L et al [22] proposed a data placement algorithm based on Lagrangian relaxation, which considers massive data transfer between cross-domains and the load of distributed systems in different geographical locations. They developed an optimal data placement model in solving data transmission cost, data center capacity constraints, and load balancing constraints. By transforming the bandwidth cost problem into a multi-source shortest path problem, the optimal data placement scheme is determined using linear programming and Lagrangian relaxation algorithms. However, this data placement algorithm may not be adaptable enough to dynamically changing network topologies.

In summary, traditional scheduling methods do not fully consider the impact of task communication dependency across domains, and fail to assign tasks based on the characteristics of cross-domain cluster environments. Current research in cross-domain scheduling faces limitations, such as inadequate division and integration of communication-dependent tasks, and insufficiently flexible and efficiency in handling latency beyond acceptable standards. To address these issues, we propose Td-Stream, a strategy aimed at more effectively managing communication dependencies, reducing latency, and enabling flexible resource adjustments for stable operation.

III. Problem Statement

This section begins with an analysis of the task dependency problem in a cross-domain computing environment, followed by task allocation and the construction of our optimization model.

A. Task Dependency

Stream computing systems need to address the challenges posed by large volumes of mixed data and complex computations. The instantaneous and non-uniform inflow of data complicates resource prediction and often leads to unbalanced resource allocation. Insufficient resources and stream congestion can slow down processing and negatively impact performance. To optimize performance, both resource allocation and task dependency must be considered to ensure that the execution order aligns with available resources.

Stream computing systems use operator instantiation to enhance resource scalability and distribute input load. The parallelization strategy splits complex computational tasks that need to be processed into independent subtasks, which are then executed in parallel on different processing units. Each operator instance operates independently, processing fragments of a data stream across processors, threads, or compute nodes.

As shown in Fig. 1, the data source generates input data that is passed to the Splitter component. The Splitter component divides the input data into multiple parts and passes these parts in parallel to different instances of a downstream operator. Each instance independently processes the data it receives and performs specific computation or operation. After being processed by individual instances, the data is aggregated at the Merger component, where the results are combined and passed to the Data Sink, the data receiver. We use

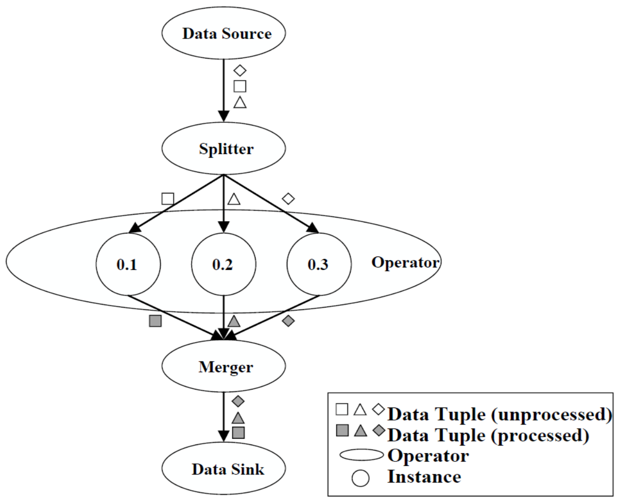

Fig. 1. Operator instantiation

Storm as an example. In Storm systems, multiple instances of spouts (data source) and bolts (data processing logic) can be processed in parallel to improve performance. Users set the number of executors to determine the degree of parallelism, and users can increase the elastic scheduling resources of component executors when congestion occurs. Topology components are configured for parallel during construction but can not be modified during execution. However, the runtime topology can be re-balanced to maintain parallelism.

In general, task dependencies are constraints on data transfer and execution order between individual tasks. The communication dependencies among tasks have a great impact on the scheduling of stream computing. Scheduling strategies need to rationally arrange the execution order of tasks according to their communication dependencies to ensure both the correctness and efficiency of the system.

In stream computing systems, the execution order of tasks is not strictly governed by dependence constraints, as component instances are relatively independent, and the execution order of upstream and downstream has little impact on response time. Therefore, in a streaming environment task dependencies primarily concern data delivery, i.e., communication dependencies.

B. Optimization Model Construction

To address the aforementioned issues, we construct three models, including a stream topology model to describe the structure of a stream application, a task dependency model to reveal inter-task dependencies and communication requirements, and a resource cost model to quantify the cost of resources required to execute tasks.

Stream topology model. Stream applications are translated into topologies. A topology has two views: logical view and physical view. The logical view consists of operators and stream. Operators are self-contained processing units responsible for performing specific operations, such as filtering or labeling. Streams represent unbounded sequences of data, such as tuples. Each operator performs partial computation on the tuples it receives and sends the partial processing results to downstream operators. The physical view is represented by a directed acyclic graph $G = (V(G), E(G))$, where $V(G)$ is a set of vertices consisting of data sources, operators, and receivers, and $E(G)$ is a set of edges along which streams flow between vertices. In this way, streaming applications enable the flow and processing of data. Each operator in a streaming application has one or more tasks, where each task is an execution instance of the operator. Each task in the operator performs the same computation on a different stream of data, thus providing parallelism. The number of tasks can be changed during runtime in response to state changes of the system.

Task dependency model. Task dependency does not consider the order of tasks execution, focusing only on the data transfer between individual tasks. Communication patterns between tasks can be categorized into inter-node and intra-node, where the communication overhead across nodes is much greater than that within nodes, and the communication overhead across processes within nodes is also greater than that in threads within processes. In order to optimize the communication pattern, the topology of tasks with dependencies is divided and the instances are allocated following the principle that the sum of task dependency degrees within a node is maximized and the sum of task dependency degrees between nodes is minimized.

Resource cost model. Each Operator instance has distinct business logic and resource requirements. Complex computations differ from simple computations in terms of resource consumption. Even within the same component, data flow characteristics or grouping patterns can affect computational overhead and lead to varying levels of resource consumption.

In a cluster, the resources of a node can be measured in dimensions such as CPU and memory. Define a resource set R consisting of l fully interconnected nodes, where $R = \{r_1, \ldots, r_l\}$. The CPU resource occupancy of node i is denoted as r_i^C. At a certain point in time, there may be multiple task instances running at node i. Let $vr_{v_{j,m},i}$ denote the amount of computational resources consumed by the task $v_{v_{j,m}}$ at node i. The computational resources consumed by task $v_{v_{j,m}}$, $vr_{v_{j,m},i}$ can be calculated by scaling the resource occupancy of the resource nodes collected during the topology runtime. It is represented by Eq. 1 and Eq. 2, where $\delta_{jn,i}$ represents the ratio between the number of data tuples transferred by this instance and the number of data tuples transferred by all tasks of node i, as represented by Eq. 3. α is the weighted value of CPU resources.

$$vr^C_{v_{j,m},i} = r_i^C \cdot \delta_{v_{j,m},i} \tag{1}$$

$$vr^M_{v_{j,m},i} = r_i^M \cdot \delta_{v_{j,m},i} \tag{2}$$

$$\delta_{v_{j,m},i} = \frac{tp_{v_{j,m}}}{\sum_{v_{k,p} \in r_i} tp_{v_{k,p}}} \tag{3}$$

$$vr_{v_{j,m}} = vr_{v_{j,m},i} = \alpha \cdot vr^C_{v_{j,m},i} + (1-\alpha) \cdot vr^M_{v_{j,m},i} \tag{4}$$

The resource overhead for a single subgraph is calculated by Eq. 5

$$f_r(g_i) = \sum_{v_{j,m} \in g_i} vr_{v_{j,m}} \tag{5}$$

IV. Task Dependency-Aware Scheduling Strategy

Building on the models mentioned above, we propose a task dependency-aware scheduling strategy, Td-Stream.

A. Td-Stream Overview

Using the topology graph partition method, cross-domain task allocation, and resource elasticity model, the cross-domain stream computing system can be optimized by considering the task dependency degree, cross-domain transmission, and resource provision. Fig. 2 shows an overview of Td-Stream's architecture on top of Storm. It includes Nimbus, Zookeeper, Supervisor, Monitoring module, database, and core algorithms of Td-Stream.

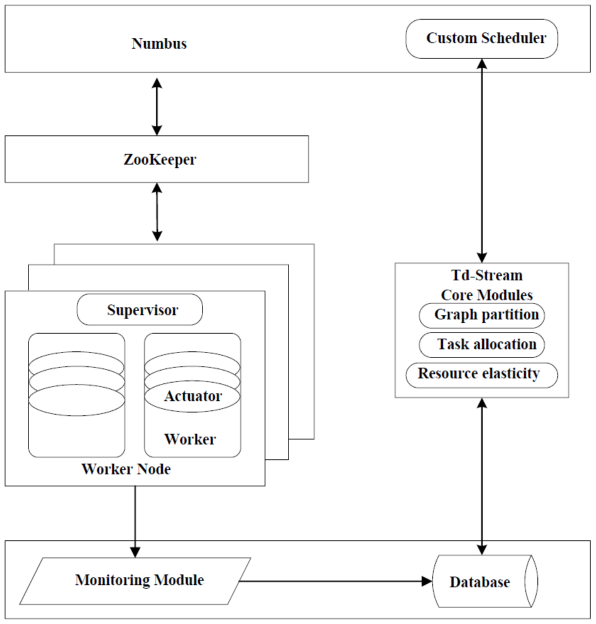

Fig. 2. Td-Stream architecture

Among them, Nimbus is responsible for coordinating and managing the operation of the entire Storm system.

Zookeeper provides distributed coordination services, maintaining the system's critical configuration information. It ensures that all components can accurately obtain the required parameters, thus guaranteeing the consistency and stable operation of the entire system. Additionally, Zookeeper unifies naming conventions, simplifying the location and access of resources in a distributed environment.

Supervisor within the compute nodes receives task assignments from Nimbus and initiates or terminates working processes accordingly. The presence of a Supervisor enables compute nodes to adapt flexibly to changes in tasks, ensuring their smooth execution.

To effectively obtain real-time data and perform topology map partitioning, task allocation and resource elasticity adjustment, we add three new modules: monitoring, database and core scheduling.

Monitoring module collects CPU load and data stream information in real-time, assessing system performance and resource requirements.

Database module stores task information and monitoring data, providing historical data support for resource allocation.

Core scheduling module is the keystone of our improvement. It utilizes Storm's IScheduler interface to replace the default scheduling algorithm. By combining real-time monitoring with historical data, it makes more precise scheduling decisions, optimizing cluster performance and response speed.

These improvements enable the Storm cluster to achieve real-time monitoring, data storage, and intelligent scheduling, better adapting to complex and dynamic data processing needs while enhancing overall system efficiency and responsiveness.

B. Cross-domain Task Allocation

Dependent tasks are distributed to nodes based on the principle of maximizing resource usage, while also considering the characteristics of clusters with cross-domain distribution. The structure of cross-domain task distribution is shown in Fig. 3.

Task allocation to nodes follows the principle of maximum resource limited allocation, taking into account the characteristics of cross-domain clusters. Priority is given to allocating tasks in the domain where the master node $DC1$ is located. Once the task

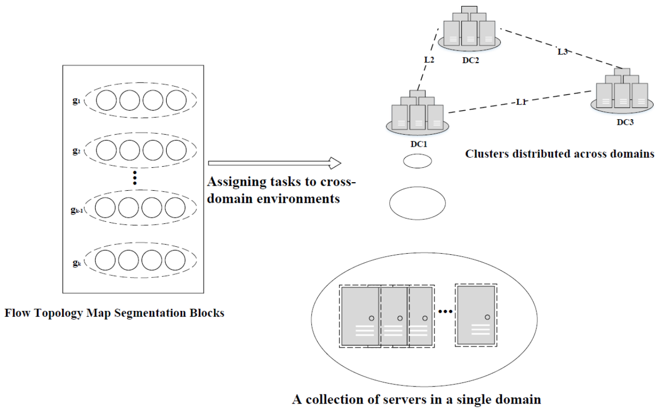
DC2
L2
L3
L1
DC1
DC3
Clusters distributed across domains
Assigning tasks to cross-domain environments
g_1
g_2
g_{k-1}
g_k
Flow Topology Map Segmentation Blocks
A collection of servers in a single domain

Fig. 3. Structure for cross-domain task allocation

volume of a node reaches a certain threshold, tasks are then assigned to other nodes within the same domain to reduce cross-domain communication dependencies. If resources in a domain are exhausted and there are still tasks unassigned, the next domain is selected. In a streaming system, task scheduling begins at the master node, making it the ideal choice to start allocation from the master node's domain. Cross-domain task allocation only occurs when there are no available resources in the master node's domain.

Cross-domain communication follows the Round-Trip Delay (RTD) model [23], considering factors such as communication latency, cost, server cost, and path reliability. To meet the low-latency requirements of stream computing, this method primarily considers the propagation time of tasks to domains. Servers are ranked based on propagation time, and tasks are allocated to domain servers one by one.

Eq. 6 represents the task dependency degree between two partitions.

$$e_{g_i,g_j} = f_{ds}(g_i, g_j) \tag{6}$$

Algorithm 1 describes the task dependency-aware allocation algorithm in a cross-domain environment.

Algorithm 1 Task Dependency Allocation Strategy in Cross-Domain Environment

```
Input: The optimal partitioning result of stream application programs $P_{\text{optimal}}$
Output: Data stream task scheduling based on task dependency
 1: Get $g_1, \cdots, g_k$ according to $P_{\text{optimal}}$
 2: Get the task dependencies between each partition block and sort them from largest to smallest: $Eg = \{e_{g_1,g_2}, e_{g_1,g_3}, \ldots, e_{g_{k-1},g_k}\}$
 3: Get the resource utilization of each node: $R_c = \{r_{c_1}, \ldots, r_{c_n}\}$
 4: Arrange the resource nodes in ascending order of different regions and resource utilization: $R = \{r_1, \ldots, r_n\}$
 5: if $G =$ null $||R =$ null then
 6:     return null
 7: end if
 8: Initialize counter $j = 1$
 9: while $G$ != null do
10:     Define or create a partition block to resource node mapping map
11:     for each resource node $r_j \in R$ do
12:         Allocate the two partition blocks connected by the largest $f_{ds}(g_A, g_B)$ to the resource node $r_j$
13:         Delete $g_A$ and $g_B$ from $G$
14:         Delete $e_{g_A,g_B}$ from Eg
15:         Update $rc_j$ {update the resource utilization of $rc_j$ }
16:         while $rc_j \leq \mu$ $||G$ != null do
17:             Allocate $g_c$ to the resource node $r_j$ {$g_c$ is the partition block connected to the largest edge weight in the partition blocks yet to be allocated }
18:             Delete $g_c$ from $G$ {Delete the partition block from the diagram}
19:             Delete $e_{g_{r_j},g_C}$ from Eg {Delete already used connecting edges from the set of connecting edges}
20:         end while
21:         Record $map_j$
22:         Map $\leftarrow map_j$ {Add the resource node mapping to the map collection }
23:         $j = j + 1$
24:     end for
25: end while
26: return the task dependency scheduling
```

Algorithm 1 inputs the optimal partitioning scheme and outputs the task allocation strategy of task dependency-aware scheduling. According to $P_{optimal}$, all the cut edges$e_{g_1,g_2}, e_{g_1,g_3}, \ldots, e_{g_{n-1},g_n}$ of the partition blocks $G = \{g_1, \ldots, g_k\}$ are sorted in descending order of the cut edge weight values (step 1), and the resource utilization rate $Rc = \{rc_1, \ldots, rc_n\}$ of each node is obtained (step 2). Domains are arranged first according to the propagation distance in different regions, then the resource nodes within each domain are sorted in ascending order of the resource utilization rate of the resource nodes, and the sorted resource nodes are stored in $R = \{r_1, \ldots, r_n\}$ (step 3). Such an arrangement satisfies the principle of reducing cross-regional data transmission while also satisfying the principle of maximum resource priority allocation. Then it is judged whether to proceed with the subsequent task allocation. If there are no partition blocks and no resource nodes, the algorithm returns a null value (steps 5 - 7).

A counter is initialized to label the number of resource nodes (step 8). When the set of partition blocks is not empty, the two partition blocks connected by the first cut edge of the sorted cut edges are assigned to the r_j resource node, and the allocated partition blocks are deleted from the set. At the same time, the cut edge is deleted and the resource utilization rate of the resource node is updated. The partition block connected by the edge with the largest adjacent edge weight is also assigned to the working node and the partition block, and the cut edge are deleted from the set until the resource node threshold is exceeded or the partition block is empty. Record each mapping method map_j until the partition block is empty, and complete the mapping Map of all partition blocks to resource nodes.

V. Performance Evaluation

Td-Stream uses Apache Storm-2.4.0 and is deployed on a cluster of CentoS7. The cluster has a total of 15 compute nodes, of which 1 node acts as a Nimbus node, 2 nodes are Zookeeper nodes for coordinated communication between master and slave nodes, and the remaining 12 nodes serve as Supervisor nodes for business logic processing. To simulate the cross-domain cluster environment, the 12 Supervisor nodes are divided into 4 groups on average, and each group is set to be in the same domain, and when there is a cross-domain data transmission, it is simulated to increase their corresponding network delay. The hardware configuration of each node is a 2-core 2.67 GHz CPU processor, 2GB of memory capacity, 40GB of hard disk storage capacity, and a 100Mbps Ethernet interface card.The version information of the software configuration is as follows: Apache Zookeeper version is Apache-Zookeeper-3.5.10, the JDK version is JDK-8u131-linux-x64, the Python version is Python-3.10.6, and the MySQL version is MySQL-5.7.

The bench-marking streaming application WordCount is used as the test topology in the comparison experiments of grouping algorithms. Its topology is shown in Fig. 4.

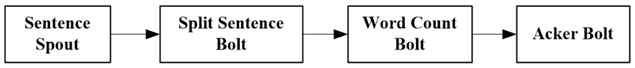

Fig. 4. WordCount topology

We compares Td-Stream strategy with DefaultScheduler, the default Storm scheduling algorithm, and Lc-Stream [24], a similar work from another recent study on cross-domain scheduling strategy. The performance metrics involved are latency, throughput.

A. Latency

System latency serves as a direct indicator of the effectiveness of the scheduling strategy employed. A short latency translates to faster processing and instant feedback. By statistically analyzing latency, we can assess the merits and drawbacks of different scheduling strategies. For this experiment, we rely on Storm UI for observations and utilize a Python crawler to record data. Below is a summary of the key points from this latency experiment.

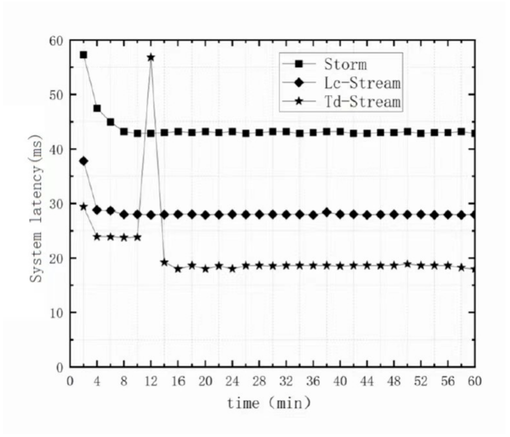

Fig. 5. System Latency under Td-Stream, Lc-Stream and Storm

As shown in Fig. 5, the total system latency over 60 minutes is presented under the three scheduling strategies: Storm, Lc-Stream, and Td-Stream. To avoid significant fluctuations in data within a single 10-second interval that may hinder observation results, the average system latency is calculated for every 120 seconds, spanning a total of 60 minutes. The total system latency is computed as the sum of the execution delay and processing delay across all components.

From Fig. 5, it can be observed that when the total system latency stabilizes, the system latency for Storm remains at around 43 ms, while the total system latency of Lc-Stream maintains at approximately 28 ms. Prior to 12 minutes, the stable total system latency for Td-Stream is around 24 ms. Upon triggering the resource elasticity mechanism, the system initiates resource optimization by adjusting the executor resource allocation for eligible compute nodes, essentially altering the number of threads allocated to components. This strategy temporarily increases the system's average latency to approximately 58 ms during its execution.

Following the execution of the strategy, the stable total system latency for Td-Stream decreases from the original 24 ms to approximately 19 ms. Compared to Storm, Td-Stream reduces the total system latency by approximately 56.9%, and compared to Lc-Stream, Td-Stream achieves a reduction of approximately 34.0%.

B. Throughput

System throughput reflects a system's data processing capability per unit of time. In this experiment, we quantify the throughput under Td-Stream, Stormstrategies. After submitting WordCount topology to the cluster, we observe through the Storm UI, and record data using a Python crawler. The experiment duration is 60 minutes, where system throughputs are measured every 300 seconds. The experimental results are shown in Figs. 6.

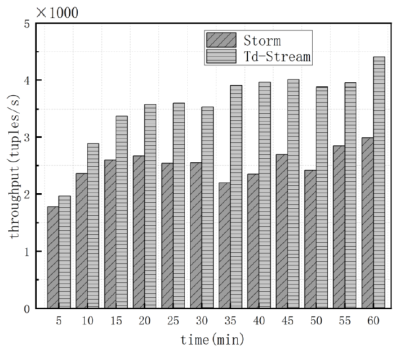

Fig. 6. WordCount throughput under Td-Stream and Storm

Fig. 6 presents the system throughputs of WordCount over 60 minutes under Td-Stream and Storm's default scheduling strategy. In Fig. 7, the system throughput is calculated as the ratio of the change in Acked tuples to the time window for the counting component of WordCount, representing the number of tuples successfully processed by the counting component per unit of time. In the first 5 minutes, due to the loading of configuration files, the system throughputs for both scheduling strategies are relatively low, below 2000 tuples/s. After 5 minutes, the system under Storm reaches a stable state with an average throughput of 2565 tuples/s. At 700 seconds, Td-Stream triggers its resource elasticity mechanism, resulting in a noticeable increase in average system throughput from 10 minutes to 15 minutes. Within 60 minutes, the average system throughput of Td-Stream is 3589 tuples/s, which represents a 39.9% improvement compared to Storm.

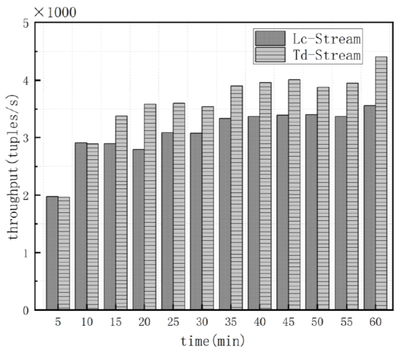

Fig. 7. WordCount throughput under Td-Stream and Lc-Stream

Fig. 7 depicts the system throughput of the WordCount over 60 minutes under Td-Stream and Lc-Stream. Td-Stream achieves an average system throughput of 3589 tuples/s, while Lc-Stream achieves an average throughput of 3200 tuples/s. Compared to Lc-Stream, Td-Stream enhances WordCount's throughput by 12.2%.

In summary, in terms of throughput performance testing, Td-Stream improves throughput by 39.3% and 11.7%, respectively, compared to Storm's default scheduling strategy and Lc-Stream.

The above experimental evaluation shows that the delay, throughputof Td-Stream strategy are significantly better than Storm default scheduling in cross-domain streaming computing environment.

VI. Conclusion and Future Work

Aiming at cross-domain streaming computing environments, the Td-Stream scheduling strategy proposed in this paper reduces the cross-domain IO transmission through fine graph division. It introduces a resource elasticity mechanism to adjust the parallelism, which effectively reduces the system delay and improves the throughput and resource utilization. Experiments show that the system performance is significantly improved under Td-Stream. However, Td-Stream scheduling has some limitations in practical applications. Firstly, it mainly focuses on data streams in regular stream computing environments but real-time data streams often have significant fluctuations. Secondly, load balancing is not considered crucial in the resource allocation process and there is no clear method to achieve load balancing by allocating tasks reasonably to fully utilize system resources while ensuring communication efficiency.

In light of these limitations,future research will focus on optimizing the data stream monitoring module for dynamic adaptation to real-time data volatility, ensuring stable system operation. Additionally, exploring algorithms for global load balancing while maintaining communication efficiency is necessary to address the current emphasis on reducing communication costs.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No.62372419; the Fundamental Research Funds for the Central Universities, China under Grant No.265QZ2021001; Melbourne-Chindia Cloud Computing (MC3) Research Network, Australia.

References

[1] Eskandari, Leila, Zhiyi Huang, and David Eyers. ”P-scheduler: adaptive hierarchical scheduling in Apache Storm.” In The Australasian Computer Science Week Multiconference. Association for Computing Machinery, 2016, pp. 1–10.

[2] Neumeyer, Leonardo. Robbins, Bruce. Nair, Anish. ”S4: Distributed Stream Computing Platform.” In 2010 10th IEEE International Conference on Data Mining Workshops. IEEE, 2010, pp. 170–177.

[3] Apache. ”Samza” [EB/OL]. http://samza.apache.org/. Accessed date 10 October 2023.

[4] Peng, B, Hosseini, M, Hong, Z, et al. ”R-Storm: Resource-Aware Scheduling in Storm.” In Proceedings of the 16th Annual Middleware Conference, 2015, 15(1), pp. 149–161.

[5] Apache. ”Storm [EB/OL]. http://storm.apache/org/. Date of access 10 October 2023.

[6] Apache. ”Flink” [EB/OL]. https://github.com/apache/flink.Accessed date 10 October 2023.

[7] Li, Ziyang. Yu, Jiong. Bian, Chen, et al. ”Flink-ER: An Elastic Resource-Scheduling Strategy for Processing Fluctuating Mobile Stream Data on Flink.” Mobile Information Systems, 2020, 2020(27), pp. 1–17.

[8] Lu, P., Zhang, L., Liu, X., Yao, J., Zhu, Z. (2015). Highly efficient data migration and backup for big data applications in elastic optical inter-data-center networks. IEEE Network, 29(5), pp. 36-42.

[9] Li, Chunlin, Cai, Qiangian, Luo, Youlong. ”Data balancing-based intermediate data partitioning and check point-based cache recovery in Spark environment.” The Joumal of Supercomputing. 2021, 78(3), pp. 3561–3604.

[10] Nasir, M.A., Morales, G.D., Garcia-Soriano, D., et al. ”The power of both choices: Practical load balancing for distributed stream processing engines.” In 2015 IEEE 31st International Conference on Data Engineering. IEEE. 2015, pp. 137–148.

[11] Guanghui, Chang. Peizhen, Li, Guangxia, Xu. ”A Highly efficient Fault Tolerance Method for An Scalable Stream Processing System.” In Proceedings of the 2017 2nd International Conference on Control, Automation and Artificial Intelligence (CAAI 2017). ATLANTIS PRESS, 2017, pp. 226–230.

[12] Liu, Xunyun, Rajkumar Buyya. ”Resource management and scheduling in distributed stream processing systems: a taxonomy, review, and future directions.” ACM Computing Surveys 53.3,2020, pp.1-41.

[13] Navroop, Kaur, Sandeep, K. Sood. ”Dynamic resource allocation for big data streams based on data characteristics (5 V s).” International Journal of Network Management, 2017, 27(4), e1978.

[14] Chunlin, L, Qianqian, C, Luo, Y. ”Optimal data placement strategy considering capacity limitation and load balancing in geographically distributed cloud.” Future Generation Computer Systems, 2021, 127,pp. 142–159.

[15] Xu, X, Li, W, Qi, H, et al. ”Latency-Constrained Cost-Minimized Request Allocation for Geo-Distributed Cloud Services.” IEEE Open Journal of the Communications Society, 2020, 1, pp. 125–132.

[16] Eskandari Leila, Mair Jason, Huang Zhiyi, Eyers David. “I-Scheduler: Iterative scheduling for distributed stream processing systems.” Future Generation Computer Systems, 2021,117, pp. 219–233.

[17] Fan Liu, Weilin Zhu, Weimin Mu, et al. “Elastic Resource Allocation Based on Dynamic Perception of Operator Influence Domain in Distributed Stream Processing.” In Computational Science – ICCC 2022,2022,13350, pp. 734–748.

[18] Liu, Y, Gao, C, Zhang, Z, et al. ”Solving NP-Hard Problems with Physarum -Based Ant Colony System.” IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2017,14(1), pp. 108–120.

[19] Peng, B., Hosseini, M., Hong, Z., et al. (2015). R-Storm: Resource-Aware Scheduling in Storm. Proceedings of the 16th Annual Middleware Conference, 15(1), pp. 149–161.

[20] Farrokh, M., Hadian, H., Sharifi, M., et al. (2022). SP-ant: An ant colony optimization based operator scheduler for high performance distributed stream processing on heterogeneous clusters. Expert Systems with Applications, 191(Suppl C), p. 116322.

[21] Hamid, H., Mohammadreza, F., Mohsen, S., et al. (2022). An elastic and traffic-aware scheduler for distributed data stream processing in heterogeneous clusters. The Journal of Supercomputing, 79(1), pp. 461–498.

[22] Chunlin L et al. “Optimal data placement strategy considering capacity limitation and load balancing in geographically distributed cloud.” Future Generation Computer Systems, 2021,127, pp. 142–159.

[23] Smith, J., and Doe, P. “Analysis of RTD in Wireless Networks.” In Proceedings of the International Conference on Wireless Communications, San Francisco, CA, USA, May 2022, pp. 250–260.

[24] Sun, D., Wang, Y., Sui, J., Gao, S., Rong, J., Buyya, R. ”Lc-Stream: An elastic scheduling strategy with latency constraints in geodistributed stream computing environments.” Currency and computation: Practice and Experience, 2024, e8085, pp. 1–22.