Rethinking elastic online scheduling of big data streaming applications over high velocity continuous data streams

Dawei Sun[1, 2], Hongbin Yan[1], Shang Gao[3], Xunyun Liu[2], and Rajkumar Buyya[2]

Abstract. Online scheduling plays a key role for big data streaming applications in a big data stream computing environment, as the arrival rate of high velocity continuous data stream might fluctuate over time. In this paper, an elastic online scheduling framework for big data streaming applications (E-Stream) is proposed, exhibiting the following features. (1) Profile mathematical relationships between system response time, multiple application fairness, and online features of high velocity continuous stream. (2) Scale out or scale in a data stream graph by quantifying computation and communication cost, and the vertex semantics for arrival rate of data stream, and adjust the degree of parallelism of vertices in the graph. Sub-graph is further constructed to minimize data dependencies among the sub-graphs. (3) Elastically schedule a graph by a priority based earliest finish time first online scheduling strategy, and schedule multiple graphs by a max-min fairness strategy. (4) Evaluate the low system response time and acceptable applications fairness objectives in a real-world big data stream computing environment. Experimental results conclusively demonstrate that the proposed E-Stream provides better system response time and applications fairness compared to the existing Storm framework.

Dawei Sun (Corresponding Author)
sundaweicn@cugb.edu.cn

Hongbin Yan
yanhongbin@cugb.edu.cn

Shang Gao
shang.gao@deakin.edu.au

Xunyun Liu
xunyunliu@gmail.com

Rajkumar Buyya
rbuyya@unimelb.edu.au

[1] School of Information Engineering, China University of Geosciences, Beijing, 100083, P.R. China.
[2] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia
[3] School of Information Technology, Deakin University, Victoria 3216, Australia

Keywords: elastic scheduling; data stream graph, streaming application; high velocity stream; big data computing.

1 Introduction

In big data era, big data stream computing helps organizations spot opportunities and risks from real time big data. It can be employed in many different application scenarios, such as social networks, trading, emergency response, fraud detection, system monitoring, smart cities, and to name but a few. More than 30000 gigabytes of data are created every second and the rate is accelerating [1]. Big data stream has some distinctive characteristics [2]. A big data stream computing system doesn't rely on high-volume storage to achieve extremely low-latency velocities. Nearly all data in a big data environment streamed. Stream computing has appeared to solve the dilemma of big data computing by processing data online within real time constraints. It makes the research on stream computing models a new trend for high-throughput computing in big data era, with both opportunities and challenges [3] [4].

In a big data stream computing environment, each application is commonly modeled as a set of sub-tasks interconnected via data dependencies, described by a corresponding DAG [2] [5] (directed acyclic graph, data stream graph, graph, DAG, and application are interchangeably used thereafter in this paper). Each DAG is submitted to a big data stream computing platform, and is scheduled to one or many computing nodes in data center. A schedule is a process of scheduling inter-dependent sub-tasks onto available computing nodes so that a DAG is able to complete its execution within specified constraints such as throughput and deadline. All the submitted applications are running continuously on the big data stream computing platform. Each application processes one or many continuous data streams. Arrival rates of data streams fluctuate over time in an unpredictable manner.

To effectively use resources, a fundamental requirement is elasticity. The majority of state-of-the-art solutions [6] [7] do not provide a proper elastic online scheduler that knows how to coordinate the dynamical allocation and release of resources according to current data stream for multiple applications. Previous work in this area, focused mostly on the static scheduling. The reason behind this is that the volume of data stream is not so big, and the magnitude of dynamically changing data steam is not so high. Many scheduling strategies provide an efficient scheduling in static stream computing environments. However, they require permanent peak-load resource provisioning to remain low latency in face of varying and busty data stream in big data era, and may cause poor resources utilization, and instability of the system as a whole. In this sense, an elastic online scheduling is always needed to avoid wasting resources or failing in delivering correct results on time.

An elastic runtime scaling strategy should be able to determine when and how to scale and account for data stream fluctuating with time, and to schedule resources elastically according to the current arrival rate of stream. To achieve that goal, we need firstly obtain a clear picture of the changed status of a graph of streaming application and then decide how to optimize it, and which vertices of the graph needed to be online rescheduled. More importantly, to achieve the scheduling fairness of multiple applications [8] [9]. Currently, most of the existing research works have focused on application scheduling. They have not considered requirements of multiple application scheduling and online features of high velocity continuous streams, nor have they sufficiently investigated how to minimize system response time and guarantee applications fairness, and to deal with high performance and response time trade-off efficiently and effectively [10] [11]. This creates the need for investigation on an elastic online scheduling framework over high velocity continuous data streams. To overcome this limitation, we propose an elastic online scheduling framework for big data streaming applications (E-Stream). It minimizes system response time, guarantees application fairness, and achieves high elasticity in a big data stream computing environment.

1.1 Observations

It is the users' responsibility to design the data stream graph in order to run a streaming application in Storm platform. However, most of the users do not possess the expertise of designing a data stream graph that reasonability reflects the performance requirement and resource consumption of the application. Key parameters such as operator parallelism and task allocation are hard to determine and optimize in an online environment where the remaining resources and rates of data stream are constantly changing over time. Besides, users have limited knowledge about the runtime behavior of the application prior to the submission, therefore, the data stream graph statically designed at compile time may eventually lead to resource over-utilization or under-utilization without delivering satisfactory performance.

However, there are few techniques available in the middleware level to optimize a submitted application. When a data stream graph is submitted, its structure is detected and optimized by the following strategies: vertex separation, fusion, and replicate. If the load calculation of a vertex is significantly higher than that of other vertices, it normally indicates that it is difficult to assign appropriate resources to this vertex. If this is the case, this vertex is separated into two or more vertices. When the traffic between two directly connected vertices is obviously greater than that of other communication links, it means that the communication delay of this line will be greater than other links, two vertices are then fused into one vertex, to eliminate communication delay of this link. In running phase, the structure of data stream graph is adjusted through vertex replication or elimination. When the input rate of data stream becomes higher, it means that latency of some critical vertices increases. One or more vertices of a group of critical vertices are replicated. When the input rate of data stream becomes lower, it means that latency of some critical vertices decreases, and some resource can be released. One or more vertices of a group of critical vertices are eliminated given those critical vertices have more than one replicas.

In an online scheduling environment, optimize the structure of data stream graph is always required. Multiple applications are sharing computing nodes in a data center so that scheduling fairness needs to be guaranteed.

1.2 Key contributions

Our contributions made in this paper are summarized as follows:

(1) Formal definitions of data stream graph, optimizing the structure of a data stream graph by quantifying and adjusting the degree of parallelism of vertices in the graph.

(2) Sub-graph is further constructed to minimize data dependencies among the sub-graphs.

(3) Data stream graph is scheduled with a priority based earliest finish time first elastic online scheduling strategy to minimize system response time.

(4) Multiple graphs are scheduled with a max-min fairness based multiple DAGs scheduling strategy to guarantee fairness subject to the constraint of response time.

(5) Prototype implementation and performance evaluation of the proposed E-Stream, which makes trade-off between low system response time and acceptable applications fairness objectives efficiently and effectively.

1.3 Paper organization

The rest of this paper is organized as follows: In section 2, the related work on workflow scheduling in distributed systems, and application scheduling on Storm platform are reviewed. Section 3 present the data stream graph model, multiple user model, data center model and multiple data stream graph scheduling model are presented. Section 4 focuses on the computation and communication cost, vertex semantics, instance of vertices, sub-graph construction, single DAG scheduling, and multiple DAG scheduling in the proposed E-Stream framework. Section 5 provides the experimental environment, parameter setup and performance evaluation of E-Stream. Finally, conclusions and future work are given in section 6.

2 Related Work

In this section, two broad categories of related work are presented: workflow scheduling in distributed systems, and application scheduling on Storm platform.

2.1 Workflow scheduling in distributed systems

Workflow scheduling problem in distributed systems is scheduling the dependent vertices of workflow on the available computing nodes of the distributed systems to satisfy the user's specified SLAs constraints such as deadline. Finding an optimal schedule for precedence constraint based directed acyclic graph is proved to be NP-hard. It has been studied extensively over the years, and will continue to be the focus of research due to its theoretical significance and practical importance.

In [12], a cloud-aware scheduling system is designed. The system has two sub-systems, a sub-system will separate a graph into multi sub-graphs, and another sub-system will allocate those sub-graphs to a cluster according to load balancing strategy.

In [13], an analytical cost model is constructed. The workflow scheduling problem is formulated as an optimization problem. A recursive critical path based work-flow scheduling is proposed, a rigorous workflow analysis is designed, and a layer oriented programming strategy is developed.

In [14], a dynamic workflow scheduling strategy is proposed. The strategy focused on scheduling resources for precedence constraints tasks to a datacenter, and the deadline is one of the major considering factors.

In [15], a budget constrained allocation approach is proposed. The approach can guarantee the cost in the specified budget, and minimizes the deadline of work-flow.

In [16], an integrated solution for workflow scheduling is proposed. The workflow scheduling problem is formulated. The integrated solution try to minimize the end-to-end delay of workflow.

To summarize, the aforementioned solutions provide a valuable insight into the challenges and potential solutions for application scheduling in big data stream computing environments. However, in big data era, novel approaches that address the particular challenges and opportunities of these technologies need to be developed, and some characteristics specific to big data stream computing environments need to be considered when developing online scheduling strategies.

2.2 Application scheduling on Storm platform

In big data era, Storm is the most popular big data stream computing platform both in academia and industry. On Storm platform, the round-robin scheduling is employed. It is simplistic and un-intelligent, in which many of the basic factors are not considered, such as, throughput performance, resource availability, or resource demands and availability. Some works have been done to improve the application scheduling strategy on Storm platform.

In [1], an adaptive scheduling approach for Storm platform is proposed. The transfer rate and traffic pattern of data stream are considered in the approach. The number of required resources can be obtained by the proposed approach, and can also be adaptively refreshed.

In [7], a dynamic resource scheduling strategy for cloud based data stream system is proposed. It includes an accurate performance model, and can process application topologies.

In [8], a resource aware scheduling mechanism is proposed in Storm platform, and to maximize resource utilization while minimizing network latency. Hard constraints and soft constraints are considered in the mechanism.

In [17], a stream data computing strategy is designed for Storm platform. The traffic aware scheduling approach can minimize inter-node and inter-process traffic. The fine grained control approach can achieve improved system performance.

In [18], an online scheduling strategy for Storm platform is proposed. The topology structure is analyzed in the offline environment, and the performance monitoring is employed in the online environment, and is used in the rescheduling stage.

In [19], an elastic scheduling framework named CE-Storm is designed. The framework can scale-out and scale-in of Continuous Query operators. Data provider can also design the specifically confidentiality policies.

In [20], a GPU-enabled parallel system is proposed for Storm platform. The system exposes GPUs to Storm applications.

In [21], a set of improvements to a distributed stream computational model is provided. The extensions of Storm platform are designed.

Additionally, our past work [2] focused on masking failures of computing nodes and communication links in streaming computing environments, we proposed a fault tolerant framework for streaming computing platform to improve the system reliability. In this paper, we focus on the fairness of multiple graphs scheduling in streaming computing environments. Another past work [25] of our group focused on improving system stability in streaming computing environments, and we proposed a stable online scheduling strategy for forever online applications. In this paper, however, our primary goal is not stability but elasticity. We propose an elastic online scheduling framework for multiple online applications, which minimizes system response time, guarantees application fairness, and achieves high elasticity in big data stream computing environments.

To summarize, current application scheduling on Storm platform are limited to one or other aspects. Up to now, most of the research required permanent peak-load resource provisioning to maintain low latency in face of varying and busty data streams, which may cause not only poor resources utilization but also instability of the system as a whole. In this sense, an elastic online scheduling for big data streaming applications is always needed. It is necessary to have an elastic online scheduler, to scale out or scale in the application to avoid wasting resources or failing to deliver correct results on time.

3 Problem Statement

To precisely reflect elastic online scheduling problem, we present the data stream graph model, the multiple user model, the datacenter model, and the multiple data stream graph scheduling model.

3.1 Data stream graph model

A big data stream application is usually described by a data stream graph G, composed of vertices set and directed edges set. It has a logical structure and specific function, and denoted as $G=\left(V(G),E(G)\right)$, where $V(G)=\{v_1,v_2,\cdots,v_n\}$ is a finite set of n vertices. $E(G)=\{e_{1,2},e_{1,3},\cdots,e_{n-i,n}\}\subset V(G)\times V(G)$ is a finite set of directed edges. The logical structure of a data stream graph G is usually described by DAG [22] [23]. Each big data stream application has a deadline associated with it. A deadline is defined as time limit for the execution of the application [24].

The makespan M of G is the total elapsed time required to execute G. For simplicity, the makespan M can be set to a value equal to the early finish time EFT_{v_e}

of the end vertex v_e, and is also equal to the latest finish time LFT_{v_e} of the end vertex v_e, as shown in (1), more details can be found in [25].

$$M = EFT_{v_e} = LFT_{v_e}. \tag{1}$$

3.2 Multiple user model

Elastic online application scheduling system typically consists of multiple users [25]. Let $U = \{u_1, u_2, \cdots, u_m\}$ be a user set composed of m users, $Gs = \{Gs_1, Gs_2, \cdots, Gs_m\}$ be a set of data stream graphs of the user set U. For simplicity, it is assumed that a user always has only one application (described by a data stream graph) at any time.

Multiple users share resource in a data center. For each user, the available resource with elastic strategy is always needed. For all users, fair resource allocation is always needed.

3.3 Datacenter model

A data center DC is usually described as an undirected graph, composed of a computing node set and undirected edge set. It has a physical structure and specific functions, as shown in Figure 1, more details of data center DC can be found in [25].

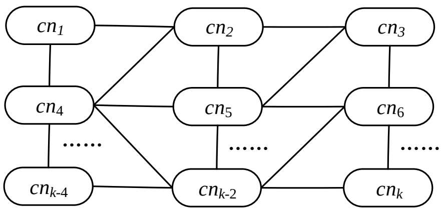

Fig. 1. A data center

3.4 Multiple data stream graph scheduling model

In an online scheduling environment, we focus on finding an elastic scheduling strategy to optimize the execution of multiple data stream graphs on a set of shared computing nodes, and maximize the system fair-ness with makespan guaranteed.

A fair multiple DAG scheduling strategy mean that resources allocation is the same with that in non-shared allocation environment [24] [26] [27].

For a DAG g_i, total allocated resources $Tar_{g_i}(t_k)$ in [0, t_k] is the accumulated resources, as shown in (2).

$$Tar_{g_i}(t_k) = \int_0^{t_k} ar_{g_i}(t)dt, \quad \textbf{(2)}$$

where $ar_{g_i}(t)$ is the current allocated resources for DAG g_i at time *t*.

The total needed allocated resources $Tnr_{g_i}(t_k)$ in [0, t_k] is the accumulated resources, as shown in (3).

$$Tnr_{g_i}(t_k) = \int_0^{t_k} nr_{g_i}(t)dt, \quad \textbf{(3)}$$

where $nr_{g_i}(t)$ is the current needed resources for DAG g_i at time *t*.

The fairness degree $fd_{g_i}(t_k)$ for DAG g_i at time t_k is defined in (4).

$$fd_{g_i}(t_k) = \frac{Tar_{g_i}(t_k)}{Tnr_{g_i}(t_k)} = \frac{\int_0^{t_k} ar_{g_i}(t)dt}{\int_0^{t_k} nr_{g_i}(t)dt}, \quad \textbf{(4)}$$

As total actual allocated resources $Tar_{g_i}(t_k)$ is always no more than total needed resources $Tnr_{g_i}(t_k)$, so $fd_{g_i}(t_k) \in [0,1]$. If $fd_{g_i}(t_k) = 1$, it implies the absolute resource fairness for DAG g_i at time t_k, all the needed resources are allocated. If $fd_{g_i}(t_k) = 0$, it implies the absolute resource un-fairness for DAG g_i at time t_k, none of the needed resources is allocated. The greater the fairness degree $fd_{g_i}(t_k)$ for DAG g_i at time t_k, the more fairness the share resources in data center.

For all *n* DAGs, fairness degree $Fd_{ng}(t_k)$ for *n* DAGs at time t_k is the average of all *n* DAGs, is defined in (5).

$$Fd_{ng}(t_k) = \frac{1}{n}\sum_{i=1}^{n} fd_{g_i}(t_k), fd_{g_i}(t_k) \in [0,1], \quad \textbf{(5)}$$

For a good fairness strategy, it should be able to maximize $Fd_{ng}(t_k)$. The proposed data stream graph scheduling model is defined by Definition 1.

Definition 1: *Data stream graph scheduling model*. In a big data stream computing system, let the data stream graph scheduling model *Gm* be represented by a four-tuple $Gm = (U, DC, Of, \Theta)$, where $U = \{u_1, u_2, \cdots, u_m\}$ is a user set composed of *m* users, and each user may request services independently. $DC = \{cn_1, cn_2, \cdots, cn_n\}$ be a data center composed of *n* computing nodes, which are running on virtual machines or physical machines. For each data stream graph, *Of* is an objective function to schedule each data stream graph. It is defined according to (6), and Θ is an algorithm which implements optimal strategies to minimize the makespan with guaranteed system fairness.

$$Of\left(avg\left(m(G)\right),Fd_{ng}\left(t_k\right)\right)=\min\left(avg\left(m(G)\right)\middle|Fd_{ng}\left(t_k\right)\right),$$
$$s.t.\ avg\left(m(G)\right)\le\delta,Fd_{ng}\left(t_k\right)\in[0,1]. \tag{6}$$

4 E-Stream Overview

In order to provide a bird's-eye view of the elastic online scheduling framework E-Stream, in this section, we discuss the overall structure of the E-Stream, which includes computation and communication cost, vertex semantics, instance of vertices, sub-graph construction, single DAG scheduling, and multiple DAG scheduling.

4.1 Computation and communication cost

Computation cost [28] c_{v_i,cn_j} is the time required to run vertex v_i on computing node cn_j, and is related to the instructions number n_{instr,v_i} of the tasks in vertex v_i, and processing ability p_{cn_j} of computing note cn_j.

Communication cost [29] $c_{e_{i,j}}$ of directed edge $e_{i,j}$ is the time required to transmit data tuple from vertex v_i to v_j, and is related to the data output d_{v_i} of vertex v_i, bandwidth $b_{e_{i,j}}$ of the directed edge $e_{i,j}$. Specifically, if v_i and v_j run on the same computing node, then $c_{e_{i,j}}=0$.

We refer to reference [25] for more detailed discussion on the computation and communication cost.

4.2 Vertex semantics

The semantic of vertex v_i [30] [31] in data stream graph G indicates relationships between input stream I_{v_i} and output stream O_{v_i} of vertex v_i, which is $O_{v_i}=F_{v_i}\left(I_{v_i}\right)$. The semantic of vertex v_i can be further classified into 4 types, as shown in Figure 2.

(1) 1:1 type

In the 1:1 type, as shown in Figure 2(a), there are one input stream I and one output stream O of vertex v_i, ir_{v_i} is the rate of input stream I, or_{v_i} is the rate of output stream O. ir_{v_i} and or_{v_i} are related with time complex degree of the tasks in vertex v_i and processing ability p_{cn_j} of the computing node cn_j, which are constants. For simplicity, the relationship of ir_{v_i} and or_{v_i} can be described as (7).

$$or_{v_i} = \alpha_I \cdot ir_{v_i} + \beta_I, \alpha_I, \beta_I \in (0, +\infty), \quad (7)$$

where α_I, β_I are the scaling factors describing the scaling out or scaling in of ir_{v_i} and or_{v_i}, determined by the function of vertex v_i, and available computing power of computer node running vertex v_i.

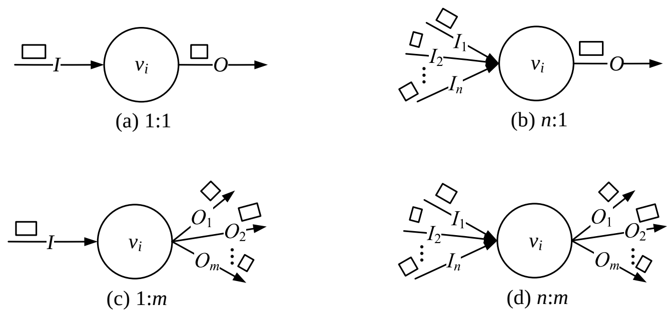

Fig. 2. Vertex semantics

(2) n:1 type

In the n:1 type, as shown in Figure 2(b), there are n input streams $I_{v_i,1}, I_{v_i,2}, \cdots, I_{v_i,n}$, and one output stream O of vertex v_i. $ir_{v_i,1}, ir_{v_i,2}, \cdots, ir_{v_i,n}$ are the rates of input streams $I_{v_i,1}, I_{v_i,2}, \cdots, I_{v_i,n}$, respectively. or_{v_i} is the rate of output stream O. The relationship between $ir_{v_i,1}, ir_{v_i,2}, \cdots, ir_{v_i,n}$ and or_{v_i} can be described as (8).

$$or_{v_i} = \sum_{k=1}^{n} \left(\alpha_{I_k} \cdot ir_{v_i,k} + \beta_{I_k} \right), \alpha_{I_k}, \beta_{I_k} \in (0, +\infty), \quad (\mathbf{8})$$

where $\alpha_{I_k}, \beta_{I_k}, k \in [1, n]$ are the scaling factors describing the scaling out or scaling in of $ir_{v_i,k}$ and or_{v_i}.

(3) 1:m type

In the 1:m type, as shown in Figure 2(c), there are one input stream I and m output streams $O_{v_i,1}, O_{v_i,2}, \cdots, O_{v_i,m}$ of vertex v_i. ir_{v_i} is the rate of input stream I, $or_{v_i,1}, or_{v_i,2}, \cdots, or_{v_i,m}$ are the rates of output streams $O_{v_i,1}, O_{v_i,2}, \cdots, O_{v_i,m}$, respectively. The relationship between ir_{v_i} and $or_{v_i,1}, or_{v_i,2}, \cdots, or_{v_i,m}$ can be described as (9).

$$\begin{cases} or_{v_i,1} = \alpha_{I_1} \cdot ir_{v_i} + \beta_{I_1}, \alpha_{I_1}, \beta_{I_1} \in (0, +\infty), \\ or_{v_i,2} = \alpha_{I_2} \cdot ir_{v_i} + \beta_{I_2}, \alpha_{I_2}, \beta_{I_2} \in (0, +\infty), \\ \vdots \\ or_{v_i,m} = \alpha_{I_m} \cdot ir_{v_i} + \beta_{I_m}, \alpha_{I_m}, \beta_{I_m} \in (0, +\infty), \end{cases} \quad (9)$$

where $\alpha_{I_j}, \beta_{I_j}, j \in [1, m]$ are the scaling factors describing the scaling out or scaling in of ir_{v_i} and $or_{v_i,j}$.

(4) *n*:*m* type

In the *n*:*m* type, as shown in Figure 2(d), there are n input streams $I_{v_i,1}, I_{v_i,2}, \cdots, I_{v_i,n}$ and m output streams $O_{v_i,1}, O_{v_i,2}, \cdots, O_{v_i,m}$ of vertex v_i. $ir_{v_i,1}, ir_{v_i,2}, \cdots, ir_{v_i,n}$ are the rates of input streams $I_{v_i,1}, I_{v_i,2}, \cdots, I_{v_i,n}$, respectively, $or_{v_i,1}, or_{v_i,2}, \cdots, or_{v_i,m}$ are the rates of output streams $O_{v_i,1}, O_{v_i,2}, \cdots, O_{v_i,m}$, respectively. The relationship between $ir_{v_i,1}, ir_{v_i,2}, \cdots, ir_{v_i,n}$ and $or_{v_i,1}, or_{v_i,2}, \cdots, or_{v_i,m}$ can be described as (10).

$$\begin{cases} or_{v_i,1} = \sum_{k=1}^{n} \left(\alpha_{I_{k,1}} \cdot ir_{v_i,k} + \beta_{I_{k,1}} \right), \alpha_{I_{k,1}}, \beta_{I_{k,1}} \in (0, +\infty), \\ or_{v_i,2} = \sum_{k=1}^{n} \left(\alpha_{I_{k,2}} \cdot ir_{v_i,k} + \beta_{I_{k,2}} \right), \alpha_{I_{k,2}}, \beta_{I_{k,2}} \in (0, +\infty), \\ \vdots \\ or_{v_i,m} = \sum_{k=1}^{n} \left(\alpha_{I_{k,m}} \cdot ir_{v_i,k} + \beta_{I_{k,m}} \right), \alpha_{I_{k,m}}, \beta_{I_{k,m}} \in (0, +\infty), \end{cases} \quad (10)$$

where $\alpha_{I_{k,j}}, \beta_{I_{k,j}}, k \in [1, n], j \in [1, m]$ are the scaling factors describing the scaling out or scaling in of $ir_{v_i,k}$ and $or_{v_i,j}$.

Theorem 1. In a big data stream computing environment, rate of data stream input to computing platform is r. For a vertex v_n in data stream graph G, the output data rate or_{v_n} of vertex v_n has a linear relationship with the input data rate r.

Proof. For a path from vertex v_1 to vertex v_n,

$$or_{v_n} = \alpha_n \cdot ir_{v_n} + \beta_n.$$

If ir_{v_i} is the input data rate of vertex v_i, $or_{v_{i-1}}$ is the output data rate of vertex v_{i-1} on that path from vertex v_1 to vertex v_n, $\omega_{i,i-1}$ is weight of data stream from vertex v_{i-1} to vertex v_i on that path from vertex v_1 to vertex v_n.

That is,

$$ir_{v_i} = \omega_{i,i-1} \cdot or_{v_{i-1}}$$

So,

$$
\begin{aligned}
or_{v_n} &= \alpha_n \cdot ir_{v_n} + \beta_n \\
&= \alpha_n \cdot \left(\omega_{n-1,n} \cdot or_{v_{n-1}}\right) + \beta_n \\
&= \alpha_n \cdot \left(\omega_{n-1,n} \cdot \prod_{k=2}^{n}\left(\alpha_{k-1} \cdot ir_{v_{k-1}} + \beta_{k-1}\right)\right) + \beta_n \\
&= \prod_{k=1}^{n}(\alpha_k) \cdot \prod_{k=2}^{n}(\omega_{k-1,k}) \cdot ir_{v_1} + \sum_{h=2}^{n}\left(\prod_{k=h}^{n}(\alpha_k) \cdot \prod_{k=h}^{n}(\omega_{k-1,k}) \cdot \beta_{h-1}\right) + \beta_n.
\end{aligned}
$$

If there are m paths from vertex v_1 to vertex v_n in data stream graph G, then,

$$
\begin{aligned}
or_{v_n} &= \alpha_n \cdot ir_{v_n} + \beta_n \\
&= \alpha_n \cdot \left(\sum_{k=1}^{id_{v_i}} \omega_{k,i} \cdot or_{v_k}\right) + \beta_n \\
&= \sum_{p=1}^{m}\left(\prod_{k=1}^{n}(\alpha_k) \cdot \prod_{k=2}^{n}(\omega_{k-1,k})\right) \cdot ir_{v_1} + \sum_{p=1}^{m}\left(\sum_{h=2}^{n}\left(\prod_{k=h}^{n}(\alpha_k) \cdot \prod_{k=h}^{n}(\omega_{k-1,k}) \cdot \beta_{h-1}\right) + \beta_n\right).
\end{aligned}
$$

If,

$$\alpha = \sum_{p=1}^{m}\left(\prod_{k=1}^{n}(\alpha_k) \cdot \prod_{k=2}^{n}(\omega_{k-1,k})\right).$$

$$\beta = \sum_{p=1}^{m}\left(\sum_{h=2}^{n}\left(\prod_{k=h}^{n}(\alpha_k) \cdot \prod_{k=h}^{n}(\omega_{k-1,k}) \cdot \beta_{h-1}\right) + \beta_n\right).$$

Then,

$$or_{v_n} = \alpha \cdot ir_{v_1} + \beta.$$

As $ir_{v_1} = r$, so,

$$or_{v_n} = \alpha \cdot r + \beta.$$

□

Similarly, the relationship between end vertex v_e of data stream graph G and the input data rate r is also linear.

4.3 Instance of vertices

Replication of vertex in a data stream graph can improve throughput. Each vertex v_i can create n different independent instances $v_{ij}, j \in (1, 2, \cdots, n)$. Instances run on different machines, and work in parallel.

The number of instances of each vertex can be determined by the number of instructions that each vertex has. More details of our vertex instance model can be found in [25].

4.4 Sub-graph construction

In a DAG, the communication cost between some vertices may be significantly longer than that of other vertices, and greatly increases the response time of the DAG. In order to reduce such kind of communication cost, a sub-graph is constructed on the related vertices. A sub-graph is defined as Definition 2.

Definition 2: *sub-graph*. A sub-graph *sub-G* of data stream graph G is the sub graph consisting of a subset of the vertices with the edges in between. For any vertices v_i and v_j in the sub-graph *sub-G* and any vertex *v* in data stream graph G, v must also be in the *sub-G* if v is on a directed path from v_i to v_j, that is $\forall v_i, v_j \in V(sub\text{-}G)$, $\forall v \in V(G)$, if $v \in V\left(p\left(v_i, v_j\right)\right)$, then $v \in V\left(p(sub\text{-}\mathrm{G})\right)$.

A sub-graph *sub-G* can be substituted by a logically equivalent vertex. Construction of a sub-graph can reduce the communication cost between related vertices, and reduce the response time of the DAG. A sub-graph will be treated as a "vertex" in the DAG scheduling phase.

For a directed edge $e_{i,j}$ from vertex v_i to v_j, the communication to computation ratio ccr_{v_i,v_j} of vertex v_i and v_j can be calculated by (11).

$$ccr_{v_i,v_j} = \frac{avg\left(c_{e_{i,j}}\right)}{avg\left(c_{v_i}\right) + avg\left(c_{v_j}\right)}. \qquad \textbf{(11)}$$

where $avg\left(c_{v_i}\right)$ is the average computation cost of vertex v_i, and $avg\left(c_{e_{i,j}}\right)$ is the average communication cost from vertex v_i to v_j.

If the communication to computation ratio ccr_{v_i,v_j} of vertex v_i and v_j meet condition (12), a sub-graph need to be constructed.

$$ccr_{v_i,v_j} > \delta, \qquad \textbf{(12)}$$

where δ is the adjust parameter, which can be set according to needs of different stream computing environments. Such as, δ can be set as 1, which means the computation cost of vertex v_i and v_j equal to the communication cost of directed edge $e_{i,j}$.

4.5 Single DAG scheduling

For a DAG, a priority based earliest finish time first scheduling strategy is employed [32].

In a DAG, each vertex can be set with a priority according to its location in the DAG. The priority of vertex v_i is defined by (13).

$$p\left(v_i\right)=\max_{\forall v_k \in set_{childre}\left(v_i\right)}\left\{p\left(v_k\right)+c_{e_{i,k}}\right\}+avg\left(c_{v_i}\right), \tag{13}$$

where v_k is one of children of vertex v_i, $set_{children}\left(v_i\right)$ is children set of vertex v_i, and $avg\left(c_{v_i}\right)$ is the average computation cost of vertex v_i.

The priority of the end vertex v_e is defined by (14).

$$p\left(v_e\right)=avg\left(c_{v_e}\right) \tag{14}$$

The priority of a vertex determines the order in which the resources are allocated. The source vertex v_s always has the highest priority among all vertices in the DAG, and it is always first scheduled to a computing node. At the beginning, all vertices in the DAG are added to a non-schedule vertices set in topological order. When a vertex is scheduled to a node, the vertex is removed from the non-schedule vertices set, and added to schedule set. A vertex is always scheduled to a computing node on which the earliest completion time is guaranteed.

The earliest finish time EFT_{v_s,cn_j} of vertex v_i running on computing node cn_j is shown in (15).

$$EFT_{v_s,cn_j}=t_{v_i,cn_j}^{idle}+c_{v_i,cn_j}. \tag{15}$$

The earliest finish time EFT_{v_s} is the finish time of source vertex v_s on computing node cn_{pbest} with minimum total time of available time and computing time, as shown in (16).

$$EFT_{v_s,cn_{Pbest}}=\min_{cn_j \in ava\left(v_i\right)}\left\{t_{v_i,cn_j}^{idle}+c_{v_i,cn_j}\right\}. \tag{16}$$

where $ava\left(v_i\right)$ is the set of available computing nodes for vertex v_i.

For other vertices in G, to calculate EST_{v_i,cn_j}, all immediate predecessor vertices of v_i must have been scheduled, and added to the schedule set.

$$EST_{v_i,cn_j}=\max\left\{t_{v_i,cn_j}^{idle}, \max_{v_{pred} \in pred\left(v_i\right)}\left\{EFT_{v_{pred}}+c_{e_{pred,i}}\right\}\right\}, \tag{17}$$

where t_{v_i,cn_j}^{idle} is the earliest time at which computing node cn_j is ready for v_i use, and $pred\left(v_i\right)$ is the set of immediate predecessor vertices of vertex v_i.

The earliest finish time EFT_{v_i,cn_j} of vertex v_i running on computing node cn_j can be calculated by (18).

$$EFT_{v_i,cn_j}=EST_{v_i,cn_j}+c_{v_i,cn_j}. \tag{18}$$

The earliest finish time EFT_{v_i} is the finish time of vertex v_i on the computing node cn_{pbest} with minimum total time of available time and computing time, as shown in (19).

$$EFT_{v_i,cn_{Pbest}} = \min_{cn_j \in ava(v_i)} \left\{ EFT_{v_i,cn_j} \right\}. \quad \textbf{(19)}$$

where $ava(v_i)$ is the set of available computing nodes for vertex v_i.

The following three rules are also employed in scheduling a DAG.

Rule 1: each instance of a vertex is scheduled to a different computing node.

If a vertex has multiple instances, each instance of the vertex is scheduled to a different computing node, to improve the efficiency of node usages. If two or more instances are schedule to the same node, it is not only unhelpful to improve the efficiency, but also increases the workload of the node.

Rule 2: the computing node with the maximum available computing power is always employed.

If a vertex can be scheduled to multiple nodes, given the same earliest finish time, the node with the maximum available computing power is always employed. As the available computing power of a node keeps changing, the most remaining available "powerful" node is not always the same. This rule helps achieve a fairer use of all available resources.

Rule 3: minimize number of vertices in the elastic online rescheduling stage.

When a DAG is scheduled on computing platform, it is running forever. If the arrival rate of data stream or the number of available computing nodes is changed, the DAG is to be rescheduled during this stage, the scheduling strategy is the same as the strategy for single DAG. However, the current allocation status is to be considered. The vertex to be scheduled on the same node will not be further rescheduled to minimize the number of vertices to be rescheduled.

4.6 Multiple DAG scheduling

For a *n*-DAGs scheduling scenario, a max-min fairness based multiple DAGs scheduling strategy is employed [33], and described as Algorithm 1.

Algorithm 1: Max-min fairness based multiple DAGs scheduling algorithm.

1. **Input**: multiple DAGs, current available capacity ability matrix $C_{v_{n\times m}}$ of computing nodes in data centers, input rate of data stream.
2. **Output**: Max-min fairness based multiple DAGs scheduling algorithm with makespan guaranteed.
3. **if** DAG *G* or computing nodes is null **then**
4. Return null.
5. **end if**
6. Monitor the real-time rate of data stream in the input interface and response time of each DAG.
7. **while** some DAGs need more resources **do**
8. Sort resources needed DAGs in ascending order by the number of

resources needed.
9. **while** set of resources needed DAGs is not null **do**
10. Select a DAG g_i needing the least resources.
11. **if** available resources in data center is greater than required **then**
12. Allocate resources for DAG g_i by priority based earliest finish time first strategy.
13. Update available capacity of the affected nodes.
14. **end if**
15. Update current available capacity matrix $C_{v_{n\times m}}$ of nodes in data centers.
16. Update the set of resources needed DAGs.
17. **end while**
18. Monitor the real-time rate of data stream in the input interface and response time of each DAG.
19. Update the set of resources needed DAGs
20. **end while**
21. **return** Max-min fairness based multiple DAGs scheduling sequence with makespan guarantee.

The input of this algorithm is multiple DAGs, current available capacity matrix $C_{v_{n\times m}}$ of computing nodes, and input rate of data stream. The output is max-min fairness based multiple DAGs scheduling sequence with makespan guaranteed. Step 7 to step 20 monitor those DAGs requiring more resources, and reschedule all those DAGs by priority based earliest finish time first strategy. The makespan is maximized with system fairness degree guaranteed.

5 Performance Evaluation

To evaluate the performance of the proposed E-Stream system, we created the experimental environment and conducted experiments as discussed below.

5.1 Experimental environment and parameter setup

Storm platform [17] [34] [35] is one of the most popular big data stream computing platforms in industry today. It is a parallel, distributed, and fault-tolerant system, designed to provide a platform that supports real-time data stream computing on clusters of horizontally scalable commodity machines.

The proposed E-Stream system is developed based on Storm 0.10.2, and installed on top of Linux Ubuntu Server 13.04. Real data experiments are performed on a computing cluster located at computer architecture laboratory in China University of Geosciences, Beijing. The computing cluster consists of 35 machines, with one designated as master node, running Storm Nimbus, two designated as Zookeeper node, and the rest 32 machines working as worker nodes. Each machine runs Linux Ubuntu Server 13.04 with dual 4-core, Intel Core (TM) i7-4790, 3.6GHz, 4 GB Memory, and 1Gbps network interface cards.

Moreover, an instance graph of TOP_N (see Figure 3), and an instance graph of WordCount (see Figure 4), are submitted to the data center.

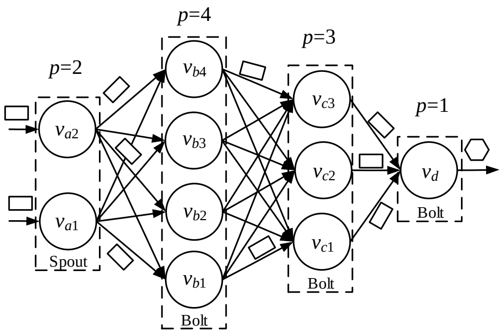

Fig. 3. Instance graph of TOP_N in Storm.

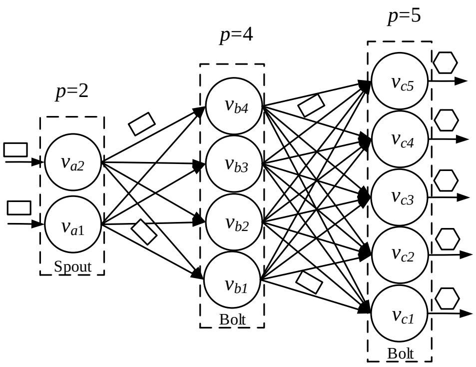

Fig. 4. Instance graph of WordCount in Storm.

5.2 Performance results

The experimental setting contains two evaluation parameters: the response time *RT*, and the fairness degree *FD*.

(1) *Response time*. The response time *RT* or makespan of a DAG is determined by the critical path of that DAG. *RT* can be calculated by EFT of the end vertex v_e. It can also be obtained from Storm UI.

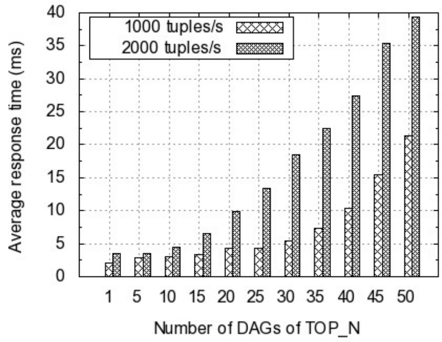

Fig. 5. Average response time of instance graph of TOP_N with different number of DAGs.

Given the rate of data stream is stable, with the increase of number of DAGs, the average response time also increases. As shown in Figure 5, when the rate of data stream set at 1000 tuples/s, and 2000 tuples/s, the average response times of instance graph of TOP_N are increasing with the number of DAGs accordingly. However, even when the number of DAGs of TOP_N is 50, the rate of data stream set at 1000 tuples/s, and 2000 tuples/s, the average response time of instance graph of TOP_N is 21.35 ms, and 39.32ms, respectively, which is reasonably acceptable in an online stream computing environment.

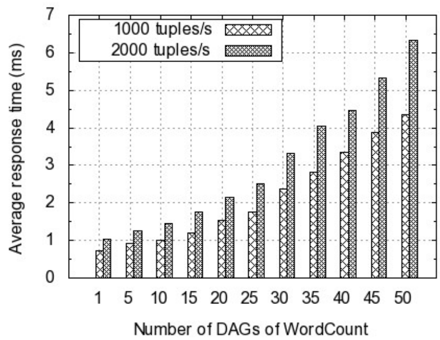

Fig. 6. Average response time of instance graph of WordCount with different number of DAGs.

Given the rate of data stream is stable, with the increase of number of DAGs, the response time of DAG also increases. As shown in Figure 6, when the rate of data stream set at 1000 tuples/s, and 2000 tuples/s, the average response times of instance graph of WordCount are also increasing with the number of DAGs accordingly. However, even when the number of DAGs of WordCount is 50, when the rate of data stream set at 1000 tuples/s, and 2000 tuples/s, the average response time of instance graph of WordCount is 4.35 ms, and 6.32ms, respectively, which are reasonably acceptable in an online stream computing environment.

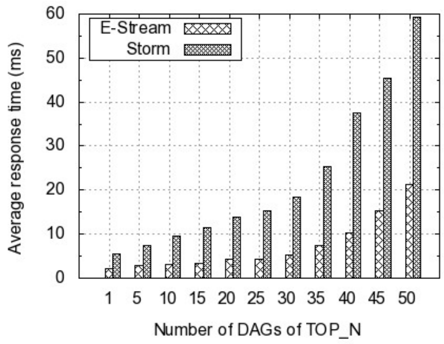

Fig. 7. Average response time of instance graph of TOP_N with data rates 1000 tuples/s.

Given the rate of data stream is stable, E-Stream has a better average response time compared with the default, round-robin strategy of Storm platform. As shown in Figure 7, with the rate set at 1000 tuples/s, the average response time of instance graph of TOP_N by E-Stream is greatly shorter than that of the default Storm strategy under the same situation. The larger number of DAGs, the higher improvement of the average response time by E-Stream.

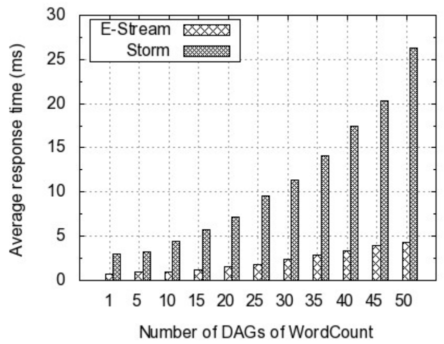

Fig. 8. Average response time of instance graph of WordCount with data rates 1000 tuples/s.

Given the rate of data stream is stable, E-Stream also has a better average response time, compared with the default round-robin strategy on Storm platform. As shown in Figure 8, with the rate set at 1000 tuples/s, the average response time of instance graph of WordCount by E-Stream is greatly shorter than that of the default Storm strategy under the same situation. The larger number of DAGs, the higher improvement of the average response time by E-Stream.

(2) *Fairness degree*. Fairness degree *FD* reflects fairness of all related DAGs in a data center. Fairness degree $Fd_{ng}\left(t_k\right)$ for *n* DAGs at time t_k is the average of all *n* DAGs, as defined in (5). If $Fd_{ng}\left(t_k\right)=1$, it implies the absolute resource fairness for

n DAGs at time t_k. If $fd_{g_i}(t_k)=0$, it implies the absolute resource un-fairness for n DAGs at time t_k. The greater the fairness degree $Fd_{ng}(t_k)$ for n DAGs at time t_k, the more fairness the sharing resources in data center.

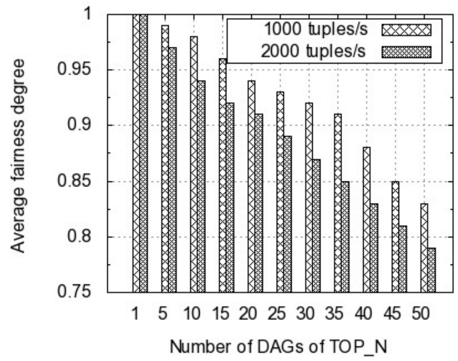

Fig. 9. Average fairness degree of instance graph of TOP_N with different number of DAGs.

Given the rate of data stream is stable, with the increase of number of DAGs, the fairness degree of all DAGs decreases. As shown in Figure 9, when the rate of data stream set at 1000 tuples/s and 2000 tuples/s, the average fairness degree of instance graph of TOP_N is decreasing with the number of DAGs. However, even when the number of DAGs of TOP_N is 50, the rate of data stream is 1000 tuples/s, and 2000 tuples/s, t_k =100s, the average fairness degree of instance graph of TOP_N is 0.83, and 0.79, respectively, which are reasonably acceptable in an online stream computing environment.

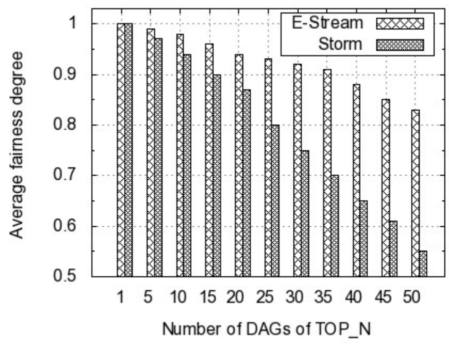

Fig. 10. Average fairness degree of instance graph of TOP_N with data rates 1000 tuples/s.

Given the rate of data stream is stable, E-Stream has a better average fairness degree, compared with the default round-robin strategy on Storm platform. As shown in Figure 10, with the rate set at 1000 tuples/s, the average fairness degree of instance graph of TOP_N by E-Stream is greatly better than that of the default strategy by Storm under the same situation. The larger number of DAGs, the higher improvement of the average fairness degree by E-Stream.

6 Conclusions and Future Work

Elastic online scheduling over high velocity continuous data streams is one of the major obstacles for opening up the new era of big data stream computing. In a big data stream computing environment, each DAG is submitted to a big data stream computing platform, and scheduled on one or many computing nodes in data center. All the submitted applications are running continuously. An elastic online scheduling is always needed to improve resource usage.

An elastic runtime scaling strategy is the key part of elastic online scheduling framework, which determines when and how to scale, and accounts for data stream fluctuating with time. A clear picture of the changed status of a graph of streaming application is firstly obtained. It is then decided how to optimize the graph of application, and which vertices of the graph need to be online rescheduled. More importantly, the scheduling fairness of multiple applications is achieved. It is investigated as to understand how to minimize system response time and guarantee applications fairness.

Our contributions made in this paper are summarized as follows:

(1) Formal definitions of data stream graph, optimizing the structure of a data stream graph by quantifying and adjusting the degree of parallelism of vertices in the graph.

(2) Sub-graph is further constructed to minimize data dependencies among them.

(3) Elastic scheduling of a graph by a priority based earliest finish time first strategy, and elastic scheduling of multiple graphs by a max-min fairness based strategy.

(4) Prototype implementation, experimental, and performance evaluation of the proposed E-Stream.

Our future work will be focusing on the following directions:

(1) Developing a complete elastic online scheduling framework based on E-Stream as a part of big data stream computing services to satisfy the low response time and high applications fairness objectives.

(2) Deploying the E-Stream on real big data stream computing environments.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No.61602428; the Fundamental Research Funds for the Central Universities under Grant No. 2652015338; and Melbourne-Chindia Cloud Computing (MC3) Research Network. We are grateful to Prof. Satish Srirama for his comments on improving the paper.

References

1. Eskandari, L., Huang, Z., Eyers D.: P-Scheduler: adaptive hierarchical scheduling in apache storm. Proc. the Australasian Computer Science Week Multiconference, ACSW 2016, No. 26. ACM Press, New York (2016)
2. Sun, D. W., Zhang, G. Y., Wu, C. W., Li, K. Q., Zheng, W. M.: Building a fault tolerant framework with deadline guarantee in big data stream computing environments. Journal of Computer and System Sciences, vol. 89, pp. 4-23. (2017)
3. Dayarathna, M., and Toyotaro, S.: Automatic optimization of stream programs via source program operator graph transformations. Distributed and Parallel Databases, vol. 31(4), pp. 543-599. (2013)
4. Alexandrov, A., Salzmann, A., Krastev, G., Katsifodimos, A., Markl V.: Emma in Action: Declarative Dataflows for Scalable Data Analysis. Proc. the 2016 International Conference on Management of Data, SIGMOD 2016, pp. 2073-2076. ACM Press, New York (2016)
5. Convolbo, M. W., Chou, J.: Cost-aware DAG scheduling algorithms for minimizing execution cost on cloud resources. The Journal of Supercomputing, vol. 72(3), pp. 985-1012. (2016)
6. Kanoun, K., Tekin, C., Atienza, D., and Shaar, M.: Big-Data Streaming Applications Scheduling Based on Staged Multi-armed Bandits. IEEE Transactions on Computers, vol. 65(12), pp. 3591-3605. (2016)
7. Fu, T. Z. J., Ding, J., Ma, R. T. B., Winslett, M., Yang, Y., Yin, Z.; Zhang, Z.: DRS: Dynamic Resource Scheduling for Real-Time Analytics over Fast Streams. Proc. 2015 IEEE 35th International Conference on Distributed Computing Systems, ICDCS 2015, pp. 411-420. IEEE Press, New York (2015)
8. Peng, B., Hosseini, M., Hong, Z., Farivar, R., and Campbell, R.: R-Storm: resource-aware scheduling in Storm. Proc. the 16th Annual Middleware Conference, Middleware 2015, pp. 149-161. ACM Press, New York (2015)
9. Choi, Y., Chang, S., Kim, Y., Lee, H., Son, W., Jin, S.: Detecting and monitoring game bots based on large-scale user-behavior log data analysis in multiplayer online games. The Journal of Supercomputing, vol. 72(9), pp. 3572-3587. (2016)
10. Lohrmann, B., Janacik, P., Kao, O.: Elastic Stream Processing with Latency Guarantees. Proc. 2015 IEEE 35th International Conference on Distributed Computing Systems, ICDCS 2015, pp. 399-410. IEEE Press, New York (2015)
11. Ahmad, S. G., Liew, C. S., Rafique, M. M., Munir, E. U., Khan, S. U.: Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems. Proc. 4th IEEE International Conference on Big Data and Cloud Computing, BDCloud 2014, pp. 129-136. IEEE Press, New York (2014)
12. Ghafarian, T., and Javadi, B.: Cloud-aware data intensive workflow scheduling on volunteer computing systems. Future Generation Computer Systems, vol. 51, pp. 87-97. (2015)
13. Gu, Y., and Wu, C. Q.: Performance analysis and optimization of distributed workflows in heterogeneous network environments. IEEE Transactions on Computers, vol. 65(4), pp. 1266-1282. (2016)
14. Chen, T. W., Lee, Y. C., Fekete, A., and Zomay, A. Y.: Adaptive multiple-workflow scheduling with task rearrangement. Journal of Supercomputing, vol. 71(4), pp. 1297-1317. (2015)
15. Arabnejad, H., and Barbosa, J. G.: A budget constrained scheduling algorithm for workflow applications. Journal of Grid Computing, vol. 12(4), pp. 665-679. (2014)
16. Yun, D., Wu, C. Q., and Gu, Y.: An integrated approach to workflow mapping and task scheduling for delay minimization in distributed environments. Journal of Parallel and Distributed Computing, vol. 84, pp. 51-64. (2015)

17. Xu, J., Chen, Z., Tang, J. and Su, S.: T-Storm: traffic-aware online scheduling in Storm. Proc. 2014 IEEE 34th Internatoin Conference on Distributed Computing Systems, ICDCS 2014, pp. 535-544. IEEE Press, New York (2014)
18. Aniello, L., Baldoni, R., Querzoni, L.: Adaptive online scheduling in Storm. Proc. the 7th ACM International Conference on Distributed Event-Based Systems, DEBS 2013, pp. 207-218. ACM Press, New York (2013)
19. Katsipoulakis, N. R., Thoma, C., Gratta, E. A., Labrinidis, A., Lee, A. J., Chrysanthis, P. K.: CE-Storm: confidential elastic processing of data streams. Proc. the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD 2015, pp. 859-864. ACM Press, New York (2015)
20. Chen, Z., Xu, J., Tang, J., Kwiat, K., and Kamhoua, C.: G-Storm: GPU-enabled high-throughput online data processing in Storm. Proc. the 2015 IEEE International Conference on Big Data, Big Data 2015, pp. 307-312. IEEE Press, New York (2015)
21. Basanta-Val, P., Fernández-García, N., Wellings, A. J., and Audsley, N. C.: Improving the predictability of distributed stream processors. Future Generation Computer Systems, vol. 52, pp. 22-36. (2015)
22. Verma, A., and Kaushal, S.: Cost-time efficient scheduling plan for execution workflows in the cloud. Journal of grid computing, vol. 13(4), pp. 495-506. (2015)
23. Gu, L., Zeng, D., Guo, S., Xiang, Y., and Hu, J.: A general communication cost optimization framework for big data stream processing in geo-distributed data centers. IEEE Transactions on Computers, vol. 65(1), pp. 19-29. (2016)
24. Tang, S., Lee, B. S., and He, B.: Fair resource allocation for data-intensive computing in the cloud. IEEE Transactions on Services Computing, doi:10.1109/TSC.2016.2531698. (2017)
25. Sun, D. W., and Huang, R.: A stable online scheduling strategy for real-time stream computing over fluctuating big data streams. IEEE Access, vol. 4, pp. 8593-8607. (2016)
26. Hu, M., Luo, J., Wang, Y., Lukasiewycz, M., and Zeng, Z.: Holistic scheduling of real-time applications in time-triggered in-vehicle networks. IEEE Transactions on Industrial Informatics, vol. 10(3), pp. 1817-1828. (2014)
27. Alkhanak, E. N., Lee, S. P., Rezaei, R., and Parizi, R. M.: Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. Journal of Systems and Software, vol. 113, pp. 1-26. (2016)
28. Hu, M., Luo, J., Wang, Y., Veeravalli, B.: Adaptive scheduling of task graphs with dynamic resilience. IEEE Transactions on Computers, vol. 66(1), pp. 17-23. (2017)
29. Matei, Z., Dhruba, B., Joydeep, S. S., Khaled, E., Scott, S., and Ion, S.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. Proc. 5th European confer ence on Computer systems, EuroSys 2010, pp. 265-278. ACM Press, New York (2010)
30. Bala, A., and Chana, I.: Intelligent failure prediction models for scientific workflows. Expert Systems with Applications, vol. 42(3), pp. 980-989. (2015)
31. Zeng, L., Veeravalli, B., and Zomaya, A. Y.: An integrated task computation and data management scheduling strategy for workflow applications in cloud environments. Journal of Network and Computer Applications, vol. 50, pp. 39-48. (2015)
32. Shi, J., Luo, J., Dong, F., Zhang, J., and Zhang, J.: Elastic resource provisioning for scientific workflow scheduling in cloud under budget and deadline constraints. Cluster Computing, vol. 19(1), pp. 167-182. (2016)
33. Zhu, Z., Zhang, G., Li, M., and Liu, X.: Evolutionary multi-objective workflow scheduling in cloud. IEEE Transactions on Parallel and Distributed Systems, vol. 27(5), pp. 1344-1357. (2016)
34. Gu, Y., and Wu, C. Q.: Performance analysis and optimization of distributed workflows in heterogeneous network environments. IEEE Transactions on Computers, vol. 65(4), pp. 1266-1282. (2016)

35. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., Ryaboy, D., "Storm@twitter," Proc. 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD 2014, pp. 147-156. ACM Press, New York (2014)

Author/s:
Sun, D;Yan, H;Gao, S;Liu, X;Buyya, R

Title:
Rethinking elastic online scheduling of big data streaming applications over high-velocity continuous data streams

Date:
2018-02-01

Citation:
Sun, D., Yan, H., Gao, S., Liu, X. & Buyya, R. (2018). Rethinking elastic online scheduling of big data streaming applications over high-velocity continuous data streams. JOURNAL OF SUPERCOMPUTING, 74 (2), pp.615-636. https://doi.org/10.1007/s11227-017-2151-2.