Journal of Computer and System Sciences 89 (2017) 4–23

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Building a fault tolerant framework with deadline guarantee in big data stream computing environments

Dawei Sun [a,b], Guangyan Zhang [a,*], Chengwen Wu [a], Keqin Li [c], Weimin Zheng [a]

[a] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, PR China
[b] School of Information Engineering, China University of Geosciences, Beijing, 100083, PR China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 14 January 2016
Received in revised form 21 October 2016
Accepted 29 October 2016
Available online 23 November 2016

Keywords:
Big data
Critical path
Deadline guarantee
Fault tolerance
Online applications
Stream computing

ABSTRACT

Big data stream computing systems should work continuously to process streams of online data. Therefore, fault tolerance is one of the key metrics of quality of service in big data stream computing. In this paper, we propose a fault tolerant framework with deadline guarantee for stream computing called FTDG. First, FTDG identifies the critical path of a data stream graph at a given data stream throughput, and quantifies the system reliability of a data stream graph. Second, FTDG allocates tasks by the fault tolerance aware heuristic and critical path scheduling mechanism. Third, FTDG online optimizes the task scheduling by reallocating the critical vertices on the critical path of the data stream graph to lower the response time and reduce system fluctuations. Theoretical as well as experimental results demonstrate that the FTDG makes a desirable trade-off between high fault tolerance and low response time objectives in big data stream computing environments.

1. Introduction

1.1. Background and motivation

Big data stream computing, the long-held dream of high-throughput computing which uses programs that compute continuous data streams, has opened up the new era of future computing due to big data, which are datasets that are too large, too fast, too dispersed, and too unstructured, and thus beyond the ability of available hardware and software facilities to undertake their acquisition, access, analysis and/or applications in a reasonable amount of time and space. Some popular features of big data are described by *n*Vs, high Volume, high Velocity, high Variety, high Veracity, high Validity, high Value, and so on. The rise of big data presents big opportunities and big challenges. Stream computing is an effective way to support big data and cloud computing by providing extremely low-latency velocities with massively parallel processing architectures, and is becoming the fastest and most efficient way to obtain useful knowledge from big data, allowing organizations to react quickly when problems appear or to predict new trends in the near future [1–4].

Big data stream computing can be employed in many different scenarios, such as stock market analysis, click streams analysis, traffic stream analysis, and emergency response, to name but a few. Usually, when compared with batch data, big data stream is difficult to be processed in real time with traditional data computing infrastructures, as it has the

* Corresponding author.
E-mail addresses: sundaweicn@cugb.edu.cn (D. Sun), gyzh@tsinghua.edu.cn (G. Zhang), wcw14@mails.tsinghua.edu.cn (C. Wu), lik@newpaltz.edu (K. Li), zwm-dcs@tsinghua.edu.cn (W. Zheng).

http://dx.doi.org/10.1016/j.jcss.2016.10.010
0022-0000/

following distinctive characteristics [2,5–7]. (1) The data are not all available at once as they arrive one tuple by one tuple in continuous data stream form. (2) The order of arrival of each data tuple cannot be controlled, when the same data tuples are to be recomputed, the order of those tuples is always different with that of before. (3) The input data stream rate is often at a high speed level and might fluctuate with time, or some statistical properties might change and increase computing and communication demands, failing to deal with such cases may result in performance bottlenecks, and at worst, data loss. So the big data stream computing system needs to online adjust and elastically adapt to the change of input data stream. (4) Timely analysis of the data stream is very important as the life cycle of most of the data is very short, all the data tuples will be processed in real time, and each data tuple can be processed only once. (5) The scale of data is infinite, and the infinite scale of data needs to be processed under tight constraints, furthermore, the volume of data is so high that there is not enough space for storage, and not all data need to be stored, so the batch computing model with the feature of store then computing is not fit all. Nearly all the data in big data environments have the feature of stream, and thus stream computing has appeared to solve the dilemma of big data computing by computing data online within real time constraints. So the stream computing model will be a new trend for high-throughput computing in big data era, and it is urgent to investigate the challenges in big data stream computing systems.

The issue of high fault tolerance is one of the major obstacles for opening up the new era of reliable stream computing in big data environments. While in most current stream computing environments, high performance and service level objectives are under consideration, high fault tolerance is ignored. In big data stream computing environments, the data centers are usually composed of tens of thousands of diverse sets of computing nodes with different capabilities and interconnected with arbitrary network architectures, failures are inevitable due to the inherently unreliable nature of the computing nodes and communication links. Many different types of failures are actually correlated with each other and have adverse effects on applications running on such environments [8,9]. So it is needed to consider fault tolerance in data centers with the expansion of the scale of the data centers for big data stream computing and ever-rising demand for higher reliability of big data stream computing applications, while maintaining high performance and service level objectives.

To achieve reliable stream computing in big data computing environments and to address high fault tolerance by carefully choosing the running computing nodes and communication links in data centers, it is important to obtain a clear picture of the total reliability of the computing nodes and communication links in big data environments. More importantly, it is needed to understand how to maximize the system reliability and minimize the response time in data centers, and to deal with the trade-off between high fault tolerance and low response time objectives efficiently and effectively, which is missing in most existing researches in big data stream computing environments [5–9]. While in most current big data stream computing environments, low response time is under consideration, high fault tolerance is ignored. This justifies the importance of modeling a fault tolerant framework with deadline guarantee for online applications in big data stream computing environments, so as to maximize the system reliability and minimize the response time to achieve high fault tolerance and low response time objectives measurement and management in big data stream computing environments.

Our work is motivated by the following observations.

(1) Existing resource scheduling strategies are inefficient.

In existing big data stream computing systems (e.g., the Storm system and the S4 system), the desirable architecture is under consideration, but highly efficient resource scheduling strategy is ignored. For example, the instance number of task topology in the Storm system is statically set by users, which cannot adapt to the changes in data stream dynamically. The round-robin strategy is employed to allocate resources when a task topology is submitted to the system. However, this has been shown to be inefficient. Obviously, a resource scheduling strategy with the feature of elasticity and adaptability to changes in data streams is needed. The authors of [10] have also tried to solve this problem. However, their solution only focuses on response time, but many factors are not considered, such as initial scheduling and critical path of task topology. In our previous work [23], we have proposed a critical path and critical vertex based resource scheduling and optimization framework, and have solved this problem in some extent. In this paper, we try to solve this problem from the perspective of system fault tolerance. According to our practice and observation, fault tolerance is more important in big data stream computing environments than that in other computing environments.

(2) The factor of high fault tolerance is not considered.

At present, high fault tolerance is not considered in designing a big data stream computing system. Most existing scheduling algorithms in big data stream computing systems only focus on response time. For example, in Storm system, fault tolerance is achieved by the timeout mechanisms (30 seconds, by default). Obviously, this fault tolerance strategy is meaningless in big data stream computing environments as the fault checking delay is too long. It is also meaningless if we only consider the response time while fault tolerance is not considered in designing a big data stream computing system. Usually, in the data centers of big data stream computing environments, the reliability of each computing nodes and communication link is different. So, high system reliability can be achieved by adjusting the distribution of the vertices running on the computing nodes.

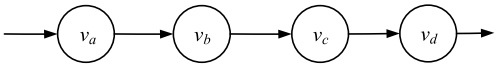
Hence, we try to achieve high fault tolerance and low response time in a big data stream computing environment. Our approach is partially inspired by the following ideas.

(1) Precedence constraint of data stream graph.

In big data stream computing environments, most of the applications are descried by data stream graph, which can be as simple as a linear pipeline or as complicated as precedence constraint based directed acyclic graph. As shown in Fig. 1(a),

```
builder.setSpout("a", new TestWordSpout(), 1); //set the number of instances of v_a
builder.setBolt("b", new RollingCountBolt(9, 3), 1).
    fieldsGrouping("a", new Fields("word")); //set the number of instances of v_b
builder.setBolt("c", new IntermediateRankingsBolt(TOP_N), 1).
    fieldsGrouping("b", new Fields("obj")); //set the number of instances of v_c
builder.setBolt("d", new TotalRankingsBolt(TOP_N), 1).
    globalGrouping("c"); //set the number of instances of v_d
```

(a)

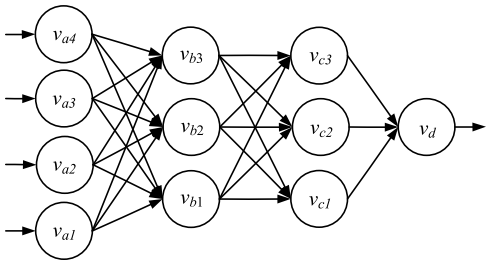


(b)

Fig. 1. a) Storm source code, and (b) data stream graph as a linear pipeline.

```
builder.setSpout("a", new TestWordSpout(), 4); //set the number of instances of v_a
builder.setBolt("b", new RollingCountBolt(9, 3), 3).
    fieldsGrouping("a", new Fields("word")); //set the number of instances of v_b
builder.setBolt("c", new IntermediateRankingsBolt(TOP_N), 3).
    fieldsGrouping("b", new Fields("obj")); //set the number of instances of v_c
builder.setBolt("d", new TotalRankingsBolt(TOP_N), 1).
    globalGrouping("c"); //set the number of instances of v_d
```

(a)



(b)

Fig. 2. (a) Storm source code, and (b) data stream graph as a precedence constraint based directed acyclic graph.

the Storm source code is the main part to achieve TOP_N computing function, the corresponding data stream graph is a linear pipeline, as shown in Fig. 1(b), where each vertex only has one instance.

As shown in Fig. 2(a), the Storm source code is also the main part to achieve TOP_N computing function, the corresponding data stream graph is a precedence constraint based directed acyclic graph, as shown in Fig. 2(b), where vertex v_a is parallelized into four instances, vertex v_b and vertex v_c are parallelized into three instances.

According to research about the scheduling precedence constraint based directed acyclic graph in distributed heterogeneous computing environments, finding an optimal schedule has been studied for years and is known to be NP-hard. Therefore, heuristics are used to obtain a suboptimal scheduling rather than parsing all possible schedules [7–9].

(2) Live migration of the vertices.

Once a data stream graph is submitted to big data stream computing environments, it will online run forever until the user kills it or some errors occur. When some factors have changed to a special extent in the big data stream computing environments, one or many vertices in the data stream graph needs to be reallocated according to a special online reallocation strategy in order to adapt to the latest changes. The capability of live migrating one or many vertices in the data stream graph makes it possible to meet high fault tolerance and low response time objectives by balancing the global loads among computing nodes and by moving one or many vertices in the data stream graph when a failure is predicted for a computing nodes, as shown in Fig. 3.

(3) Critical path in a directed acyclic graph (DAG).

In a DAG, the response time is determined by the latency of the critical path in the DAG. As shown in Fig. 4, in a real-time stream computing environment, rescheduling all vertices to improve the response time is not a wise choice, which will affect the online running of the DAG seriously. Rescheduling one or a few vertices is better than rescheduling all vertices. Rescheduling the heaviest vertex, i.e., vertex 35, is also not a good choice. The response time is determined by the latency of the critical path (10–21–25–12) of the DAG, while vertex 35 is not on the critical path. Vertex 25 is on the

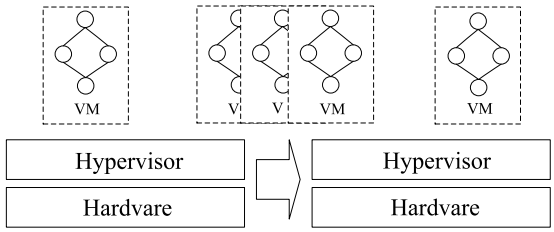


Fig. 3. Live migrating one or many vertices in the data stream graph.

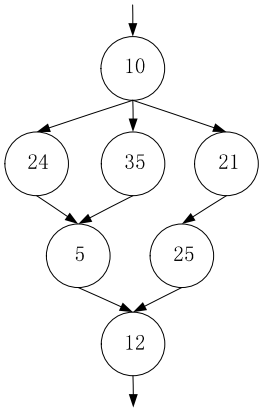


Fig. 4. Directed acyclic graph.

critical path (10–21–25–12) of the DAG. If we reschedule vertex 25, the computing time is reduced from 25 milliseconds to 20 milliseconds. Accordingly, the response time of the DAG will also be reduced by 5 milliseconds. However, if the computing time of vertex 25 is reduced form 25 milliseconds to 15 milliseconds, the response time of the DAG will be reduced by 6 milliseconds instead of by 10 milliseconds. This is because the critical path has changed and the new critical path (10–35–5–12) will determine the response time of the DAG.

All those ideas can be used to model a fault tolerant framework with deadline guarantee for online applications in big data stream computing environments, and some or all of them can be combined to use. One fact is trying to improve one aspect of the scheduling framework, it will complement each other when some or all of them are combined.

1.2. Contributions

We profile the mathematical relationship among fault tolerance, response time, and resource utilization in big data stream computing, and obtain the conditions to meet the high system reliability and low response time objectives. In this paper, we propose a fault tolerant framework with deadline guarantee for stream computing called FTDG. First, FTDG identifies the critical path of a data stream graph at a given data stream throughput, and quantifies the system reliability of a data stream graph. Second, FTDG allocates tasks by the fault tolerance aware heuristic and critical path scheduling mechanism. Third, FTDG online optimizes the task scheduling by reallocating the critical vertices on the critical path of the data stream graph to lower the response time and reduce system fluctuations.

We implement a prototype system of FTDG by modification of the Storm system. We evaluate FTDG in fault tolerance and response time in big data stream computing environments. Theoretical and experimental results demonstrate that the FTDG makes a trade-off between high fault tolerance and low response time objectives efficiently and effectively. In other words, it has the ability to provide efficient fault tolerance enhancements and obvious response time reduction.

Our contributions are summarized as follows:

(1) Formal definitions of data stream graph, big data stream computing architecture, and high system reliability and low response time objectives;

(2) Systematic investigation of the computation of nodes, communication among nodes, allocating tasks by the fault tolerance aware heuristic and critical path scheduling mechanism, and building data stream graph based reliability analysis model;

(3) Identification of the critical path of a data stream graph at a special data stream throughput, and online optimizing the scheduling mechanism;

(4) Quantification of the system reliability of a data stream graph, and consolidating non-critical vertices on non-critical path to meet high fault tolerance objective;

(5) Prototype implementation, simulation, and performance evaluation of the FTDG.

1.3. Paper organization

The remainder of this paper is organized as follows. In Section 2, definitions of data stream graph and applications scheduling model in big data stream computing systems are given. Section 3 presents the system architecture, workflow description, and algorithm description of the FTDG framework. Section 4 focuses on critical path quantification in data stream graph. Section 5 describes the failure modeling, vertex reliability quantification, and demonstrates the reliability of data stream graph from the quantitative perspective. Section 6 addresses the simulation environment, parameter setup and performance evaluation of the proposed heuristic vertex online allocation algorithms. Section 7 reviews the related works on big data stream computing, performance-oriented data stream scheduling, and fault tolerance strategy in distributed heterogeneous computing system. Finally, conclusions and future directions are given in Section 8.

2. Problem statement

In big data stream computing environments, a stream application is partitioned into a set of sub-task where in each sub-task can depend on the results of executions of other sub-task in the application, and is described by a data stream graph, which is described by directed acyclic graph (DAG). All those directed acyclic graphs will be submitted to data center. Each vertex of a directed acyclic graph is mapped to a computing node in a data center. To precisely reflect these factors in the schedule, and to optimize and evaluate the data stream graph for big data stream computing, some related definitions are given as follows [6,11–14].

2.1. Data steam graph

In stream computing, multiple continuous parallel data streams can be processed by data stream graph, and are usually represented as directed acyclic graph. A measurable view of data stream graph can be defined by Definition 1.

Definition 1 *(Data stream graph).* A data stream graph G is a directed acyclic graph composed of a vertex set and directed edge set, has a logical structure and special function, and is denoted as $G = (V(G), E(G))$, where $V(G) = \{v_1, v_2, \cdots, v_n\}$ is a finite set of n vertices, which represent sub-tasks that can be executed on any of the available computing nodes, and $E(G) = \{e_{1,2}, e_{1,3}, \cdots, e_{n-i,n}\} \subset V(G) \times V(G)$ is a finite set of directed edges, which represent the execution precedence between vertices. If $\exists e_{i,j} \in E(G)$, then $v_i, v_j \in V(G)$, $v_i \neq v_j$, and $\langle v_i, v_j \rangle$ is an ordered pair, where the data stream comes from v_i, and go to v_j. This means that v_i should complete its execution before v_j starts its execution.

The vertex v_i is an abstract computational step and is characterized by a five-tuple $v_i = (id_{v_i}, f_{v_i}, c_{v_i}, i_{v_i}, o_{v_i})$, where id_{v_i} is the identification of vertex v_i, f_{v_i} is the function of vertex v_i, c_{v_i} is the computation cost of vertex v_i, i_{v_i} is the input data stream for vertex v_i, and o_{v_i} is the output data stream for vertex v_i.

Vertex v_i represents a task composed of a set of instructions. It will change i_{v_i} by f_{v_i} to o_{v_i} with c_{v_i}, that is $o_{v_i} = f_{v_i}(i_{v_i})$. The function of data stream graph G is achieved by all the n vertices, that is $O = F(I)$, where O, F, and I are the output data steam, the function of data stream graph G, and the input data steam, respectively.

The directed edge $e_{i,j}$ is the directed communication link from vertex v_i to v_j, which is characterized by a tuple, that is $e_{i,j} = (id_{e_{i,j}}, c_{e_{i,j}})$, where $id_{e_{i,j}}$ is the identification of directed edge $e_{i,j}$, and $c_{e_{i,j}}$ is the communication cost of directed edge $e_{i,j}$ from connected vertex v_i to v_j. If v_i and v_j run on the same computing node, then $c_{e_{i,j}} = 0$.

The in-degree of vertex v_i is the number of incoming edges and the out-degree of vertex v_i is the number of outgoing edges. When all the incoming data of vertex v_i are available, it is triggered to execute. After its execution, it generates corresponding outputs, to the directly connected successor vertex (vertices) of vertex v_i. The source vertex v_s is the vertex whose in-degree is zero, and the end vertex v_e is the vertex whose out-degree is zero. A data stream graph G has at least one source vertex and one end vertex. The inputs of the source vertex come from the data source, and the outputs of the end vertex are outputted to the user. Without loss of generality, we assume that data stream graph G has exactly one source vertex v_s and one end vertex v_e. If an application has multiple source or end vertices, this data stream graph may be converted by inserting a virtual source vertex or end vertex of complexity zero connected to all source or end vertices with zero time-weight data transfer.

2.2. Application scheduling

An application scheduling system typically consists of the user broker, the scheduling broker and data centers. The scheduling broker is the data center central managing broker. Some specific features of a data stream graph scheduling system can be described as follows.

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, and $Gs_j = \{G_{j_1}, G_{j_1}, \cdots, G_{j_{m_j}}\}$ be a sub-set of data stream graphs of the jth user u_j, where m_j is the number of data stream graphs of the jth user u_j, and Gs_k is the kth data stream graph submitted to the scheduling broker through a user interface and independent of the other users. The scheduling broker schedules them to the appropriate computing nodes on the data centers. If u_0 has two data stream graphs, then $Gs_0 = \{G_{0_1}, G_{0_2}\}$, and $m_0 = 2$.

For simplicity, we assume that the data stream graphs are non-interruptible, which means that as soon as a data stream graph scheduling is started, it cannot be interrupted until all vertices in this graph is completely allocated in computing nodes in the data centers. All resources operate non-preemptively, which means that a data stream graph cannot transfer execution from one computing node to another one once it gets started.

Let $DC = \{cn_1, cn_2, \cdots, cn_n\}$ be a fully interconnected data center composed of n computing nodes, which are running in the form of virtual machines on physical machines. All those computing nodes are fully connected by an underlying communication subsystem. A data center DC can be described as an undirected graph composed of computing node set and undirected edge set, which has a physical structure and specially function, and is denoted as $UG = (CN(UG), UE(UG))$, where $CN(UG) = \{cn_1, cn_2, \cdots, cn_n\}$ is a finite set of n computing nodes, and $UE(UG) = \{ue_{1,2}, ue_{1,3}, \cdots, ue_{n-i,n}\}$ is a finite set of bi-directional communication links between the connected computing nodes. If $\exists ue_{i,j} \in UE(UG)$, then $cn_i, cn_j \in CN(UG)$, $cn_i \neq cn_j$, and (cn_i, cn_j) is a bi-directional communication link between the connected computing nodes cn_i and cn_j.

A computing node cn_k is characterized by a five-tuple $cn_k = (cid_k, cf_k, cbw_k, cp_k, cm_k)$, where cid_k, cf_k, cbw_k, cp_k, and cm_k are the computing node identification, failure probability, network bandwidth, processing capacity, and memory of computing node cn_k, respectively. The processing capacity cp_k is characterized by the frequency of CPU. cp_k can vary from $cp_k^{\min}$ to $cp_k^{\max}$ with the help of DVS, where $0 < cp_k^{\min} < cp_k^{\max}$.

The scheduling probability matrix of vertices in a data stream graph is $P_{n\times m} = \begin{pmatrix} p_{11} & \cdots & p_{1m} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nm} \end{pmatrix}$, where p_{ij} is the probability of vertex v_j in the data stream graph submitted to computing node cn_i, $p_{ij} \geq 0$, and $\sum_{i=0}^{n} p_{ij} = 1$. In order to keep computing node cn_i in serviceability state, cn_i should meet the constraint defined by (1), and the data center DC should meet the constraint defined by (2), so that tasks will not be generated in a rate exceeding the limit at which tasks can be processed.

$$\sum_{j=0}^{m} p_{ij} \cdot cdr_j \leq cp_i \cdot (1 - cf_i), \tag{1}$$

$$\sum_{i=0}^{n}\sum_{j=0}^{m} p_{ij} \cdot cdr_j \leq \sum_{i=0}^{n} cp_i \cdot (1 - cf_i). \tag{2}$$

In (1) and (2), cdr_j is the data stream arrival rate of vertex v_j on computing node cn_k.

The data stream graph scheduling model for allocating data stream graphs to the computing nodes in a data center can be defined by Definition 2.

Definition 2 *(Data stream graph scheduling model).* Let the data stream graph scheduling model Gm of a big data stream computing system 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 service independently with each other. $DC = \{cn_1, cn_2, \cdots, cn_n\}$ is a data center composed of n computing nodes, which are running in the form of virtual machines on physical machines. For each data stream graph, Of is an objective function for scheduling each data stream graph in the big data computing environment, which is defined according to (3), so as to maximize the system reliability and minimize the response time, and can be achieved by the fault tolerant framework with deadline guarantee for online applications, and Θ is an algorithm which achieves all those optimal strategies.

$$Of\big(l(G), r(G)\big) = \max\big(r(G)|l(G)\big) and \min\big(l(G)\big). \tag{3}$$

$\min(l(G))$ can be achieved by minimizing the critical path of data stream graph G, as the critical path is the longest path of G. That is,

$$\min\big(l(G)\big) = \min\big(\max\big\{l_{p_1}(v_s, v_e), \cdots, l_{p_m}(v_s, v_e)\big\}\big), \tag{4}$$

where $l_{p_i}(v_s, v_e)$ is the latencies of the ith path from vertex v_s to vertex v_e.

$\max(r(G))$ is the reliability of all those instances of the vertex set successfully running on the computing nodes with mth try strategy and all data are successfully transferred from the immediate predecessor vertex to its immediate successor vertex. It will be achieved by consolidating some vertices, not all vertices, running on non-critical path. The goal is to improve system reliability to a better degree, while avoiding too much system fluctuations. The prerequisite for those fault tolerance strategies is not affecting the minimum the response time of data stream graph G.

3. FTDG overview

In order to have a bird's-eye view of the fault tolerant framework with deadline guarantee in big data stream computing environments, in this section, we focus our attention on the overall framework of FTDG in a big data stream computing environment, which includes the system architecture, workflow description, and algorithm description [15–19].

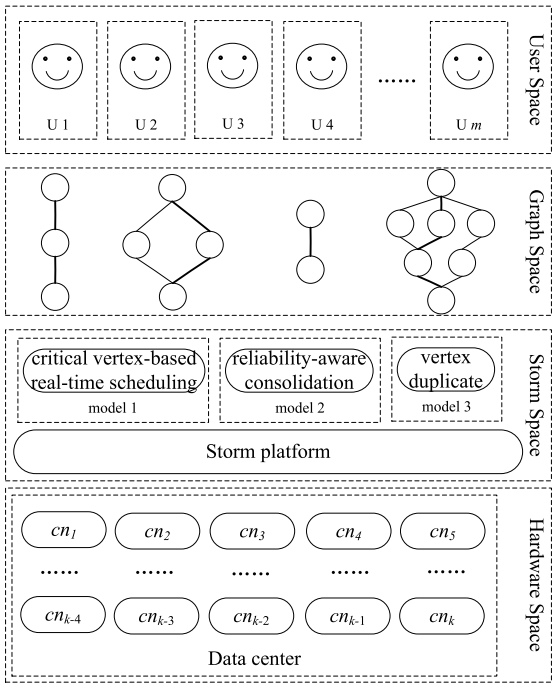


Fig. 5. FTDG system architecture.

3.1. System architecture

The system architecture of the fault tolerant framework with deadline guarantee for online applications FTDG is composed of four spaces, which are the hardware space, Storm space, graph space, and user space, as shown in Fig. 5.

In the hardware space, one or many data centers are distributed in different geographical locations. In each data center, multiple physical machines provide hardware infrastructure for big data stream computing.

In the Storm space, the critical vertex-based real-time scheduling model, reliability-aware consolidation, and vertex duplication model are employed to support real-time reliability-aware resource scheduling and optimization objectives based on the open source Storm platform [20], which is an open source and distributed big data stream computing system providing a set of general primitives for doing real time big data stream computing. The critical vertex-based real-time scheduling model is used to minimize the response time of a special data stream graph. The reliability-aware consolidation model is used to maximize system reliability of a big data stream computing environment while maintaining the response time minimized. The vertex duplication model is used to improve the response time and system reliability by creating some copies for special vertex or vertices.

In the graph space, one or many data stream graphs can be created according to the source codes, which are designed and submitted by the user. Each data stream graph represents a special user application at one time. All those data stream graphs can run concurrently.

In the user space, one or many users can design and submit applications at any time, from anywhere, by any way in the world on demand of the data center.

3.2. Workflow description

To maximize the system reliability and minimize the response time, a related definition is given in Definition 3.

Definition 3 *(Failure-aware running time, FRT).* In a big data stream computing environment, *FRT* is a running time, where the cost of failure of all the instances of the vertex set or the cost of failure of all the edges between the corresponding instances of vertices is taken into account.

FRT_{v_i} of a vertex v_i can be calculated by (5), and $FRT_{e_{i,j}}$ of a directed edge $e_{i,j}$ can be calculated by (6).

$$FRT_{v_i} = c_{v_i,cn_k} + (1 - r_{v_i}) \cdot (c_{v_i,cn_k} + \Delta t), \tag{5}$$

$$FRT_{e_{i,j}} = c_{e_{i,j}} + (1 - r_{e_{i,j}}) \cdot (c_{e_{i,j}} + \Delta t). \tag{6}$$

In (5) and (6) $c_{v_{i,k},cn_k}$ and $c_{e_{i,j}}$ are the corresponding computing time interval of vertex v_i running on computing node cn_k and the corresponding transmission time interval from vertex v_i to vertex v_j, respectively. r_{v_i} and $r_{e_{i,j}}$ are the corresponding reliability of vertex v_i, and the corresponding reliability of edge $e_{i,j}$ from vertex v_i to vertex v_j, respectively. Δt is the delay of checking a failure.

By introducing the concept of failure-aware reliable running time, FRT, the optimization objectives to maximize the system reliability and minimize the response time, can be further simplified as an objective to minimize the failure-aware reliable running time. In our work, we allocate all the vertices based on the critical path, and the vertices on the critical path will be allocated with priority allocation, taking into account vertices on the non-critical path. Under the same circumstances, when a vertex has multiple different options, the computing node or edge with minimum failure-aware reliable running time will be selected. While the failure-aware reliable running times are the same, the one with the maximum reliability will be selected.

To maximize the system reliability and minimize response time to achieve high fault tolerance and low response time objectives measurement and management in big data stream computing environments, we employ the fault tolerant framework with deadline guarantee for online applications FTDG on Storm platform. In order to achieve maximum system reliability without affecting the minimum response time, all vertices in DAG will be divided into two groups, vertices on the critical path and vertices on non-critical path. The minimize response time is only determined by all the vertices on the critical path. Furthermore, maximizing the system reliability can be achieved by consolidating the recourse for vertices without affecting the response time of DAG. Maximizing the system reliability can also be achieved by creating duplicates of vertices to make better use of the waiting time (free time) in the DAG.

In the first phase, the failure-aware running time based computation cost of each vertex and the failure-aware running time based communication cost of each edge in each DAG are estimated in the current data stream environments, the critical path and critical vertices can be calculated. When the DAG is first scheduled, a critical path based earliest effective reliable running and completion time priority scheduling strategy is employed, in which the computing cost and communication cost are under consideration. In this strategy, the vertices on the critical path will be priority scheduled to the earliest effective reliable running and completion time computing node only if its predecessors have been scheduled. If the predecessors of this vertex have not been scheduled, the predecessors will be priority scheduled to the earliest effective reliable running and completion time computing node based on a topology sort. Hereafter, when arrive rate of the data stream has changed significantly (increase or decrease), the response time can be largely improved, then a critical vertex on the critical path of a DAG is selected and scheduled to a new computing node to achieve minimum response time and system fluctuations, and the system reliability is estimated of the current resource allocation states.

In the second phase, a data stream graph reliability model is employed to estimate the system reliability in the current data stream environments. A reliability-aware consolidated non-critical vertex on non-critical path strategy is employed to maximize the system reliability while not to increases the response time of the current DAG. Usually, the more vertices in a DAG, the higher system reliability will be achieved, and the vertex duplicate strategy is used to improve the response time and system reliability by creating some copies for the special vertex or vertices, which can make better use of the free time in an allocation scheme.

3.3. Algorithm description

The algorithm of fault tolerant framework with deadline guarantee for online applications FTDG is composed of two steps, first placement of DAG and online optimization of DAG, as described in Algorithm 1. This scheduling algorithm can be implemented by rewriting the IScheduler interface of the Storm platform [20].

When a DAG is first placed to computing nodes, a critical path based earliest completion time priority placement strategy is employed, in which failure-aware running time based computing cost and communication cost are considered. In this strategy, the vertices on the critical path will be placed, in a high priority, onto the earliest completion computing node only if its predecessors have been placed. If the predecessors of this vertex have not been placed, the predecessors will be prior placed to the earliest completion computing node based on a topology sort.

When the arrival rate of the data stream has changed, the response time may also has changed. A critical vertex on the critical path of a DAG is selected and scheduled to a new computing node to achieve minimum response time and system fluctuations. A reliability quantification model of the data stream graph is employed to estimate the reliability of each DAG in the current data stream environments. A reliability-aware consolidation of non-critical vertices on non-critical path strategy is employed to maximize the system reliability without disturbing the response time of the current DAG. The vertex duplication strategy is used to improve the response time and system reliability by creating some copies for special vertex or vertices, which can make better use of the free time in an allocation scheme.

The inputs of FTDG algorithm are DAG G, capacity ability matrix $C_{v_{n\times m}}$ of the computing nodes in data centers, reliability vector R_{cn} of the computing nodes in data centers, and the data rate of the data stream in real-time. The output is a fault tolerance allocation strategy with deadline guarantee. Steps 4–21 schedule vertices to the computing nodes by critical path based earliest completion time priority scheduling strategy, in which failure-aware running time based computing cost and communication cost are under consideration. Steps 23–35 adjust in real-time the vertices scheduling strategy according to changes in the rate of the data stream to maximize the system reliability and minimize the response time, so as to achieve trade-off between high fault tolerance and low response time objectives in big data stream computing environments.

Algorithm 1 FTDG algorithm.

Input: DAG G, capacity ability matrix $C_{v_{n\times m}}$ of computing nodes in data centers, reliability vector R_{cn} of computing nodes in data centers, and data rate of data stream in real-time.
Output: A fault tolerance allocation strategy with deadline guarantee.

if DAG G or computing nodes is null **then**
  return null.
end if
Evaluate the most common data rate in a big data stream computing environment.
Calculate the failure-aware running time based computation cost of each vertex running on each computing node and the relevant failure-aware running time based communication cost of each edge between corresponding vertices, getting the computation cost matrix $C_{v_{n\times m}}$.
Determine the reliability of each computing node.
Sort all vertices topologically in DAG G.
Calculate the earliest start time EST and the latest start time LST of each vertex in DAG G.
Determine the critical path CP DAG G.
for each vertex on critical path CP of DAG G **do**
  if vertex v_i with the feature of in-degree is zero **then**
    Select vertex v_i as the will-be-selected vertex v_{sel}.
  else
    Select an immediate predecessor vertex of vertex v_i as the will-be-selected vertex v_{sel}.
  end if
  Calculate the earliest finish time EFT_{v_i} of vertex v_i on all remaining available computing nodes.
  Schedule vertex v_{sel} to the earliest finish computing node cn_j, which has the best reliability if multiple vertices meeting the time constraint.
  Refresh the capacity ability of computing nodes cn_j, and in-degree of the immediate successor vertex of vertex v_{sel}.
  Recalculate the earliest start time EST and the latest start time LST of each vertex in DAG G.
  Refresh the critical path CP of the DAG.
end for
Monitor the real-time rate of the data stream in the input interface.
while $|r_{new} - r_{old}| > \Delta r$ **do**
  Recalculate the failure-aware running time based computation cost of each vertex running on each node under the new data rate and relevant failure-aware running time based communication cost of each edge between corresponding two vertices, getting the computation cost matrix $C_{v_{n\times m}}$.
  Calculate the earliest start time EST and the latest start time LST of each vertex in DAG G and determining the critical path CP DAG G.
  Try to reschedule vertex v_i to computing nodes of the immediate predecessor or successor vertex with the constraint of remaining available capacity ability is enough, get the best improvement in response time, and recalculate critical vertex $cv_{\max}$ to the computing node $cn_{\max}$.
  Identify those computing nodes on non-critical path as the reliability-improve computing nodes.
  if one or some reliability-improve computing nodes can improve the system reliability **then**
    Schedule all vertices running on reliability-improve computing node cn to the corresponding computing nodes running the immediate predecessor or successor vertex with the constraint that the remaining available capacity ability is enough
  end if
  Recalculate the response time of DAG and the system reliability of the data center.
  if some free time in this allocation scheme **then**
    Create a copy of the nearest vertex, which is on the critical path and the EFT is earlier than the end time of the free time, to improve the response time and system reliability to an extent.
  end if
end while
return The fault tolerance allocation strategy with deadline guarantee.

4. Quantitative study of critical path

In a big data stream computing environments [15,16,21,22], for a data stream graph G, the computation cost matrix $C_{v_{n\times m}} = \begin{pmatrix} c_{v_1,cn_1} & \cdots & c_{v_1,cn_m} \\ \vdots & \ddots & \vdots \\ c_{v_n,cn_1} & \cdots & c_{v_n,cn_m} \end{pmatrix}$, is a $n \times m$ matrix, in which each computation cost c_{v_i,cn_j} (in second) is the time required when running vertex v_i on computing node cn_j. The communication cost $c_{e_{i,j}}$ (in second) of directed edge $e_{i,j}$ is the time required for transmitting the data stream from vertex v_i to v_j. If v_i and v_j run on the same computing node, then $c_{e_{i,j}} = 0$.

EST_{v_i,cn_j} and EFT_{v_i,cn_j} are the earliest start time and the earliest finish time of vertex v_i running on computing node cn_j, respectively. For the source vertex v_s, the earliest start time EST_{v_s} on any computing nodes are 0, as shown in (7), and the earliest finish time EFT_{v_s} is the finish time of running the source vertex v_s on the computing node with the best processing ability p_{cn_j}, as shown in (8).

$$EST_{v_s,cn_{any}} = 0, \tag{7}$$

$$EFT_{v_s,cn_{p_{best}}} = c_{v_i,cn_{p_{best}}}. \tag{8}$$

For the other vertices in DAG, the earliest start time can be calculated from the source vertex v_s. In order to calculate EST_{v_i,cn_j}, all immediate predecessor vertices of v_i must have been scheduled.

$$EST_{v_i,cn_j} = \max\left\{t^{idle}_{v_i,cn_j}, t^{data}_{v_{pred_1},cn_j}, t^{data}_{v_{pred_2},cn_j}, \cdots, t^{data}_{v_{pred_n},cn_j}\right\}, \tag{9}$$

where $t^{idle}_{v_i,cn_j}$ is the earliest time at which computing node cn_j is ready for use, and $t^{data}_{v_{pred_k},cn_j}$ is the time at which all input data of the kth immediate predecessor vertex of vertex v_i will be ready on computing nodes cn_j, which can be calculated by (10).

$$t^{data}_{v_{pred_k},cn_j} = EFT_{v_{pred_k}} + c_{e_{pred_k,i}}, \tag{10}$$

where v_{pred_k} is the kth immediate predecessor vertex 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 (11).

$$EFT_{v_i,cn_j} = EST_{v_i,cn_j} + c_{v_i,cn_j}. \tag{11}$$

LST_{v_i,cn_j} is the latest start time of vertex v_i running on computing node cn_j, which can be calculated by traversing the data stream graph G in a topology sort but in the reverse direction. For the end vertex v_e, the latest start time LST_{v_e,cn_j} on computing nodes cn_j is equal to the earliest start time EST_{v_e,cn_j} on computing nodes cn_j, that is.

$$LST_{v_e,cn_j} = EST_{v_e,cn_j}. \tag{12}$$

For the other vertex in DAG, the latest start time can be calculated from the end vertex v_e, where is the time at which the latest time of all output data of immediate successor vertices of vertex v_i, and can be calculated by (13).

$$LST_{v_i,cn_j} = \min_{v_{succ} \in succ(v_i)} \{LST_{v_{succ}} - c_{e_{i,succ}}\} - c_{v_i,cn_j}, \tag{13}$$

where $succ(v_i)$ is the set of immediate successor vertices of vertex v_i.

Definition 4. *Critical path, CP*, also called the longest path, is a path having the longest latencies from source vertex v_s to end vertex v_e in data stream graph G. All vertices on CP with the feature of the earliest start time EST equal to the latest start time LST.

The response time of a data stream graph G is also determined by the CP, which is equal to the earliest finish time EFT_{v_e} of the end vertex v_e.

In a big data stream computing environment, Critical Vertex, CV, is a vertex on the critical path, CP, of the data stream graph G. To the currently computing environment, if the data stream rate changes, the current allocation scheme is not a good choice, then moving CV to a new best fit computing node, the maximum response time reduction will be achieved.

The reduced time Δt_{cv} of rescheduling CV can be calculated by (14).

$$\Delta t_{cv} = \min\left\{|CP_{after} - CP_{before}|, |c_{v_{cv},cn_{after}} - c_{v_{cv},cn_{before}}|\right\}, \tag{14}$$

where CP_{after}, CP_{before}, $c_{v_{cv},cn_{after}}$, and $c_{v_{cv},cn_{before}}$ are the CP after rescheduling CV, the CP before rescheduling CV, the computation cost of CV on before rescheduling computing node cn_{before}, and the computation cost of CV on after rescheduling computing node cn_{before}, respectively.

In an online big data stream computing environment, when the data stream rate changes, the computation cost of each vertex and the communication cost of each edge will also change. Rescheduling all vertices is not a good choice, as all the data streams will be lost. The historically allocated information is not considered, and huge system fluctuations are unavoidable. In RERF framework, in the first scheduling, a critical path based earliest completion time priority scheduling strategy is employed. Hereafter, when the arrival rate of data stream has changed significantly (increase or decrease), if the response time can be largely improved, then a critical vertex on the critical path of a DAG is selected and scheduled to a new best fit computing node to achieve minimum response time and system fluctuations.

5. Reliability quantification theory

In this section, we focus our attention on the quantized system reliability in a big data stream computing environment, including failure modeling, vertex reliability quantification, and reliability quantification of data stream graphs [15,24–26].

5.1. Failure modeling

In a big data stream computing environment, the failure of a vertex or an edge will occur inevitable and will follow a special failure density function $f(t)$, which can be defined by Definition 5.

Definition 5 *(Failure density function).* A failure density function $f(t)$ of a continuous failure is a function that describes the relative likelihood for the failure to occur at a given time. The failure density function $f(t)$ is defined by (15).

$$\begin{cases} f(t) = F'(t) = \dfrac{dF(t)}{dt}, \\ s.t. f(t) \geq 0, \\ \displaystyle\int_{-\infty}^{+\infty} f(\tau)d\tau = 1, \end{cases} \tag{15}$$

where $F(t)$ is the failure distribution function, as defined in Definition 6, which is also related to the failure density function $f(t)$.

Definition 6 *(Failure distribution function).* A failure distribution function $F(t)$ of a continuous failure is a function that describes a real-valued random variable t with a given failure density function $f(t)$ and will be found at a value less than or equal to t. Intuitively, it is the "area so far" function of the failure density function $f(t)$. The failure distribution function $F(t)$ is defined by (16).

$$\begin{cases} F(t) = P(-\infty \leq t) = \displaystyle\int_{-\infty}^{t} f(\tau)d\tau, \\ 1 - F(t) = P(+\infty > t) = \displaystyle\int_{t}^{+\infty} f(\tau)d\tau, \\ F(t_b) - F(t_a) = P(t_a < t \leq t_b) = \displaystyle\int_{t_a}^{t_b} f(\tau)d\tau, \\ s.t. \lim\limits_{t\to-\infty} F(t) = 0, \\ \quad \lim\limits_{t\to+\infty} F(t) = 1. \end{cases} \tag{16}$$

Definition 7 *(Failure expectation value).* A failure expectation distribution value $E(t)$ of a continuous failure is a value that describes the weighted average of all possible failure time values that admits a probability density function $f(t)$. The failure expectation value $E(t)$ can be calculated by (17).

$$E(t) = \int_{-\infty}^{+\infty} \tau \cdot f(\tau)d\tau. \tag{17}$$

In real cases, the failure of a vertex or an edge is closely related to the physical parameters (i.e., size, material) of computing nodes or physical links. In general, the failure of a vertex or an edge is assumed to follow a Poisson distribution with constant failure rate λ. Failures of different vertices or edges are assumed to be statistically independent. Although such assumptions may not hold in reality, reasonably useful mathematical models can be obtained using this simplified assumption [9,27]. So,

$$f(x = k) = \frac{e^{-\lambda\cdot t} \cdot (\lambda \cdot t)^k}{k!}, \quad \lambda > 0,\ k \in \{0, 1, 2, 3, \cdots\}, \tag{18}$$

where $f(x = k)$ is the failure distribution function in time interval $[0, t]$.

$$F(x = k) = e^{-\lambda\cdot t} \cdot \sum_{i=0}^{\lfloor k \rfloor} \frac{(\lambda \cdot t)^i}{i!}, \tag{19}$$

$$E(t) = \lambda \cdot t, \tag{20}$$

$$\sigma(t) = \lambda \cdot t. \tag{21}$$

The reliability $r_{v_{i,k},cn_k}$ of the kth instance of vertex v_i running on computing node cn_j in time interval [0, t] is the probability of running a data tuple successfully during the time of execution, that is $k = 0$ in (18), and the fault follows a Poisson distribution with constant failure rate λ_v, then $r_{v_{i,k},cn_k}$ can be calculated by (22).

$$r_{v_{i,k},cn_k} = f(x = 0) = e^{-\lambda_v\cdot t}. \tag{22}$$

The reliability $r_{e_{i,j}}$ of the directed edge $e_{i,j}$ from vertex v_i to vertex v_j in time interval [0, t] is the probability of transferring a data tuple successfully during the time of transmission, and the fault follows a Poisson distribution with constant failure rate λ_e, then $r_{e_{i,j}}$ can be calculated by (23).

$$r_{e_{i,j}} = f(x=0) = e^{-\lambda_e \cdot t}. \tag{23}$$

5.2. Vertex reliability quantification

In big data stream computing environments, multiple instances of vertex and mth try strategy can be employed to improve the execution reliability of vertex v_i in the local level, and to provide a high fault tolerance environment of data stream graph in global level. Hence, it is important to maximize the reliability probability of each data stream graph by using the multiple instances of vertex and mth try strategy, and the vertex reliability of vertex v_i is given in Theorem 1.

Theorem 1. *In a stream computing environments, the reliability of each instance of vertex v_i running on computing node cn_j is described as $r_{v_{i,k},cn_k}$. The n instances of vertex and mth try strategy are employed to improve the reliability of vertex v_i. In the situation of where there are n instances of vertex v_i, each instance will be retried at most m times. Then, the probability that $r_{v_i,N,M\text{-}tries}$ can be calculated by (24).*

$$r_{v_i,N,M\text{-}tries} = \left(1-(1-r_{v_i,cn})^n\right) \cdot \sum_{j=1}^{m} (1-r_{v_i,cn})^{n\cdot(j-1)}. \tag{24}$$

Proof. When it comes to multiple instances of vertex, in the situation of where there are n instances of vertex v_i, and each instance of vertex v_i runs on different computing nodes. If the reliability of the kth instance of vertex v_i running on computing node cn_j is described as $r_{v_{i,k},cn_k}$, so vertex v_i is unavailable if and only if all n instances of vertex v_i are unavailable. Therefore,

$$\overline{r_{v_i,N}} = r(\overline{v_{i,1}}, \overline{v_{i,2}}, \cdots, \overline{v_{i,n}}). \tag{25}$$

Each instance of vertex v_i runs on different computing nodes, and independent of each other, thus,

$$\begin{aligned} \overline{r_{v_i,N}} &= r(\overline{v_{i,1}} \times \overline{v_{i,2}} \times \cdots \times \overline{v_{i,n}}) \\ &= r(\overline{v_{i,1}}) \times r(\overline{v_{i,2}}) \times \cdots \times r(\overline{v_{i,n}}) \\ &= \overline{r_{v_{i,1},cn_1}} \cdot \overline{r_{v_{i,2},cn_2}} \cdot \ldots \cdot \overline{r_{v_{i,n},cn_n}} \\ &= \prod_{k=1}^{n} \overline{r_{v_{i,k},cn_k}}, \end{aligned} \tag{26}$$

where $r_{v_{i,k},cn_k}$ is in interval (0, 1).

As $\overline{r_{v_{i,k},cn_k}} = 1 - r_{v_{i,k},cn_k}$, and $\overline{r_{v_i,N}} = 1 - r_{v_i,N}$, we obtain,

$$\overline{r_{v_i,N}} = \prod_{k=1}^{n} (1-r_{v_{i,k},cn_k}), \tag{27}$$

and,

$$r_{v_i,N} = 1 - \prod_{k=1}^{n} (1-r_{v_{i,k},cn_k}). \tag{28}$$

In a big data center, when the computing nodes are in virtual machine form, we can assume the reliability of each instance of vertex v_i is same and is described as $r_{v_i,cn}$, that is,

$$r_{v_{i,1},cn_1} = r_{v_{i,2},cn_2} = \cdots = r_{v_{i,k},cn_k} = r_{v_i,cn}. \tag{29}$$

Then (28) can be further simplified as (30).

$$r_{v_i,N} = 1 - (1-r_{v_i,cn})^n. \tag{30}$$

When it comes to the mth try strategy, in the situation of where there is only one instance of vertex v_i. In the mth try strategy, this strategy presumes that all the $m-1$ previous attempts have failed, and the mth try succeeds. The mth try strategy will be employed when the one instance of vertex v_i is fails in all those $m-1$ tries, then reliability $r_{v_{i,m-try},cn_k}$ of vertex v_i on computing node cn_k at the mth try can be calculated by (31).

$$\begin{aligned} r_{v_{i,m-try},cn_k} &= r(\overline{v_{i,1\text{-}th,cn_k}}, \overline{v_{i,2\text{-}th,cn_k}}, \cdots, \overline{v_{i,m-1\text{-}th,cn_k}}, v_{i,m\text{-}th,cn_k}) \\ &= r(\overline{v_{i,1\text{-}th,cn_k}} \times \overline{v_{i,2\text{-}th,cn_k}} \times \cdots \times \overline{v_{i,m-1\text{-}th,cn_k}} \times v_{i,m\text{-}th,cn_k}) \\ &= r(\overline{v_{i,1\text{-}th,cn_k}}) \times r(\overline{v_{i,2\text{-}th,cn_k}}) \times \cdots \times r(\overline{v_{i,m-1\text{-}th,cn_k}}) \times r(v_{i,m\text{-}th,cn_k}) \\ &= \overline{r_{v_{i,1\text{-}th},cn_k}} \cdot \overline{r_{v_{i,2\text{-}th},cn_k}} \cdot \ldots \cdot \overline{r_{v_{i,m-1\text{-}th},cn_k}} \cdot r_{v_{i,m\text{-}th},cn_k}. \end{aligned} \tag{31}$$

In a big data center, we can assume the reliability of one instance of vertex v_i running on computing node cn_k are same in all those m tries, and is described as r_{v_i,cn_k}, that is,

$$\overline{r_{v_{i,1\text{-}th},cn_k}} = \overline{r_{v_{i,2\text{-}th},cn_k}} = \cdots = \overline{r_{v_{i,m-1\text{-}th},cn_k}} = 1 - r_{v_{i,m\text{-}th},cn_k} = 1 - r_{v_i,cn_k}. \tag{32}$$

We obtain,

$$r_{v_{i,m\text{-}try},cn_k} = r_{v_i,cn_k} \cdot (1 - r_{v_i,cn_k})^{m-1}. \tag{33}$$

The probability that vertex v_i will be successfully completed on or before the mth try can be calculated by (34).

$$r_{v_i,M\text{-}tries,cn_k} = \sum_{j=1}^{m} r_{v_{i,j-try},cn_k} = \sum_{j=1}^{m} \left(r_{v_i,cn_k} \cdot (1 - r_{v_i,cn_k})^{j-1} \right), \tag{34}$$

which is the summation of the probability of the completion of the m tries.

When it comes to multiple instances of vertex and the mth try strategy, in the situation where there are n instances of vertex v_i and each instance of vertex v_i employs the mth try strategy when all those instance of vertex v_i in the $m-1$ previous attempts have failed, then the reliability r_{v_i} of the instance of vertex v_i at the mth try can be calculated by (35).

$$\begin{aligned} r_{v_i,N,m\text{-}th} &= r \begin{pmatrix} \overline{v_{i,1,1\text{-}th,cn_k}}, & \overline{v_{i,2,1\text{-}th,cn_k}}, & \cdots, & \overline{v_{i,n,1\text{-}th,cn_k}} \\ \overline{v_{i,1,2\text{-}th,cn_k}}, & \overline{v_{i,2,2\text{-}th,cn_k}}, & \cdots, & \overline{v_{i,n,2\text{-}th,cn_k}} \\ \vdots & \vdots & \cdots, & \vdots \\ \overline{v_{i,1,m-1\text{-}th,cn_k}}, & \overline{v_{i,2,m-1\text{-}th,cn_k}}, & \cdots, & \overline{v_{i,n,m-1\text{-}th,cn_k}} \\ \neg(\overline{v_{i,1,m\text{-}th,cn_k}}, & \overline{v_{i,2,m\text{-}th,cn_k}}, & \cdots, & \overline{v_{i,n,m\text{-}th,cn_k}}) \end{pmatrix} \\ &= \left(1 - \prod_{k=1}^{n} (1 - r_{v_{i,k},cn_k}) \right) \cdot \prod_{k=1}^{n} (1 - r_{v_{i,k},cn_k})^{m-1}. \end{aligned} \tag{35}$$

For simplicity, if the reliability of each instance of vertex v_i on each computing node are the same and is described as $r_{v_i,cn}$, then (35) can be further simplified as (36).

$$r_{v_i,N,m\text{-}th} = \left(1 - (1 - r_{v_i,cn})^n\right) \cdot (1 - r_{v_i,cn})^{n \cdot (m-1)}. \tag{36}$$

The probability that vertex v_i will be successfully completed on or before the mth try can be calculated by (37).

$$\begin{aligned} r_{v_i,N,M\text{-}tries} &= \sum_{j=1}^{m} r_{v_i,N,j\text{-}th} \\ &= \sum_{j=1}^{m} \left(\left(1 - (1 - r_{v_i,cn})^n\right) \cdot (1 - r_{v_i,cn})^{n \cdot (j-1)} \right) \\ &= \left(1 - (1 - r_{v_i,cn})^n\right) \cdot \sum_{j=1}^{m} (1 - r_{v_i,cn})^{n \cdot (j-1)}. \end{aligned} \tag{37}$$

This completes the proof. □

For a scenario, when it comes to multiple instances of vertex, if vertex v_i has three instances, and the reliability $r_{v_i,cn}$ of each instance of vertex v_i is 0.8, that is $r_{v_i,cn} = 0.8$, then reliability r_{v_i} of vertex v_i can be calculated by (30). That is $r_{v_i} = 1 - (1 - r_{v_i,cn})^n = 1 - (1 - 0.8)^3 = 0.992$. When it comes to the mth try strategy, if vertex v_i has one instance, and the instance will retry at most three times, and the reliability $r_{v_i,cn}$ of the instance of vertex v_i on computing node cn_k is also 0.8, that is $r_{v_i,cn} = 0.8$, then $r_{v_i,M\text{-}tries,cn_k} = \sum_{j=1}^{m} (r_{v_i,cn_k} \cdot (1 - r_{v_i,cn_k})^{j-1}) = \sum_{l=1}^{3} (0.8 \cdot (1 - 0.8)^{j-1}) = 0.992$. When it comes to multiple instances of vertex and the mth try strategy, if vertex v_i has three instances, and the reliability $r_{v_i,cn}$ of each instance of vertex v_i is 0.8, each instance will retry at most three times, then $r_{v_i,N,M\text{-}tries} = (1 - (1 - r_{v_i,cn})^n) \cdot \sum_{j=1}^{m} (1 - r_{v_i,cn})^{n \cdot (j-1)} = (1 - (1 - 0.8)^3) \cdot \sum_{j=1}^{3} (1 - 0.8)^{3 \cdot (j-1)} = 0.9999$.

The relationship between reliability $r_{v_i,cn}$ of each instance of vertex v_i and the reliability $r_{v_i,N}$ of vertex v_i with the n instances is shown in Fig. 6. The relationship between reliability $r_{v_i,cn}$ of each instance of vertex v_i and the reliability $r_{v_i,M\text{-}tries,cn_k}$ of vertex v_i with the m tries is shown in Fig. 7. The relationship between reliability $r_{v_i,cn}$ of each instance of vertex v_i and the reliability $r_{v_i,N,M\text{-}tries}$ of vertex v_i with the n instances and m tries is shown in Fig. 8. We can easily get the follow conclusions (1) multiple instances of a vertex in space and multiple tries of an instance in time will be able to achieve the same effect; (2) in a big data computing environment, the number of n in multiple instances and the number of m in multiple tires can be get by the reliability of each instance of a vertex and the objective reliability of the vertex; (3) as usual, when only consider the reliability of a vertex, the number of n in multiple instances should be less than 3, and the number of m in multiple tires should be less than 2.

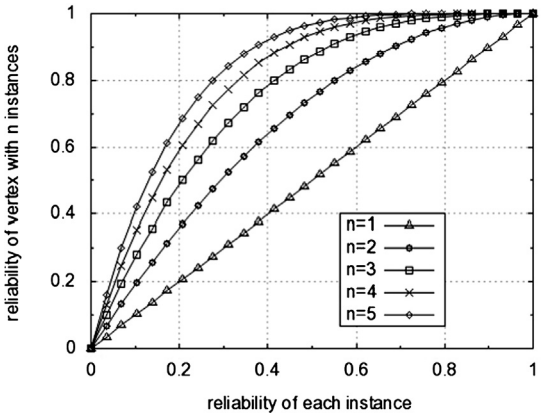


Fig. 6. Reliability of vertex with n instances.

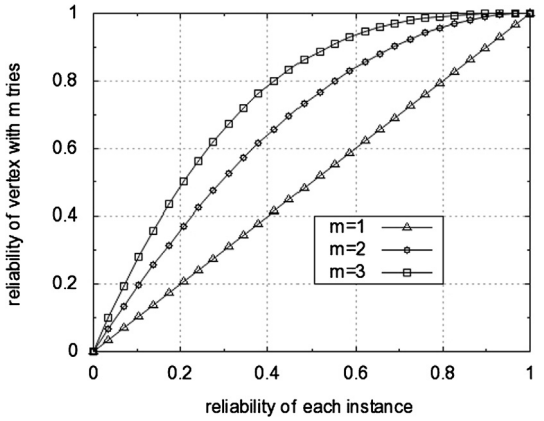


Fig. 7. Reliability of vertex with m tries.

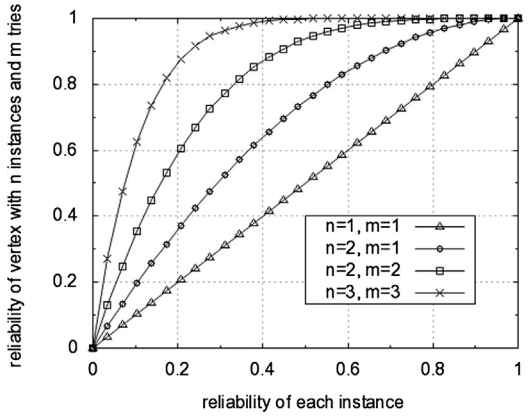


Fig. 8. Reliability of vertex with n instances and m tries.

5.3. Reliability quantification of data stream graph

Let a data stream graph $G = (V(G), E(G))$ be composed of vertex set and directed edge set, where $V(G) = \{v_1, v_2, \cdots, v_n\}$ is a finite set of n vertices and $E(G) = \{e_{1,2}, e_{1,3}, \cdots, e_{n-i,n}\}$ is a finite set of m directed edges.

Recall that failures of vertices and edges in big data stream computing environments are assumed to be statistically independent. Therefore, the $r(G)$ of data stream graph G is equal to the probability of all its instances of vertex set successfully running on computing nodes with the mth try strategy and all data are successfully transferred from its immediate predecessor vertex to its immediate successor vertex, and can be calculated by (38).

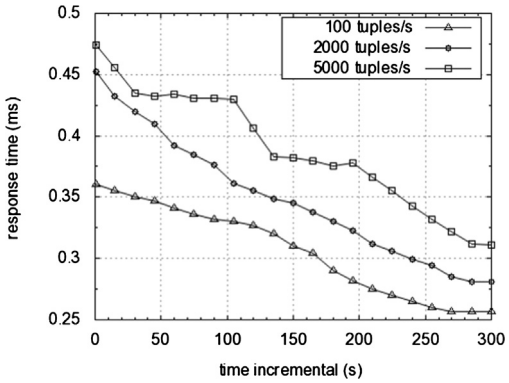


Fig. 9. Response time of DAG with different data stream rates.

$$r(G) = \prod_{i=1}^{n} r_{v_i} \cdot \prod_{e_{x,y} \in E(G)} r_{e_{x,y}}. \tag{38}$$

The reliability expectation value $E_{r(G)}[t_a, t_b]$ of data stream graph G in a corresponding computing time interval $c_{v_{i,k},cn_k}$ and a corresponding transmission time interval $c_{e_{i,j}}$ to process data stream successfully can be calculated by (39).

$$E_{r(G)}[t_a, t_b] = \prod_{i=1}^{n} r_{v_i} \cdot c_{v_{i,k},cn_k} \cdot \prod_{e_{x,y} \in E(G)} r_{e_{x,y}} \cdot c_{e_{x,y}}. \tag{39}$$

6. Simulation and performance evaluation

In order to evaluate the performance of the proposed FTDG algorithm, simulation environment and parameter set are discussed firstly in this section, followed by the precise performance evaluation results.

6.1. Simulation environment and parameter setup

Storm platform [10,20,28,29] is used in the experiment environment, which is a parallel, distributed, and fault-tolerant system. It is designed to fill the gap of providing a platform that supports real-time data stream computing on clusters of horizontally scalable commodity machines. Our experiment is done on Storm 0.8.1.

24 physical machines are created in a data center in the simulation environment. The physical machine has a dual 6-core, Intel Xeon 2 GHz, 32 bit, 4 GB Memory, and 1 Gbps network interface. Each machine runs Linux Ubuntu Server 13.04. All those machines are interconnected by arbitrary processor network, communication links are bidirectional, and its topological structure is a fully connected graph. The failure of a machine or a communication link is set to follow a Poisson distribution with constant failure rate $\lambda = 0.1$. The corresponding reliability of a machine or a communication link is usually in the range of [0.9, 1] in 1 second. Failures of different virtual machines or edges are set to be statistically independent.

Moreover, a linear pipeline based directed acyclic graph and a complicated precedence constraint based directed acyclic graph, as shown in Fig. 1 and Fig. 2, are submitted to the data centers. The function of the two DAGs is to achieve TOP_N computing in big data stream computing environments. Those two DAGs are critical path sensitive DAGs. The length of the critical path is significantly longer than other paths. The length of each tuple is considered as a random number within the range of [1000, 2000] KI.

6.2. Performance evaluation

The experimental set up contains three evaluation parameters: the response time RT, system throughput ST, system reliability $R(sys)$.

(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 of the DAG, as shown in (11). RT can also be obtained by Storm UI, which is provided by the Storm platform.

When the rate of a data stream is stable, with the increase of time, the response time of DAG will decrease. When the time is long enough, the response time will stabilized at a low level. As shown in Fig. 9, when the rate of the data stream is stable at 100 tuples/s, 2000 tuples/s, and 5000 tuples/s, the response time is 0.282 ms, 0.323 ms, and 0.378 ms, respectively.

When the rate of a data stream is stable, with the increase of the number of vertex instances, the average response time of DAG is decrease. As shown in Fig. 10, when the rate of the data stream is stable at 100 tuples/s, 2000 tuples/s,

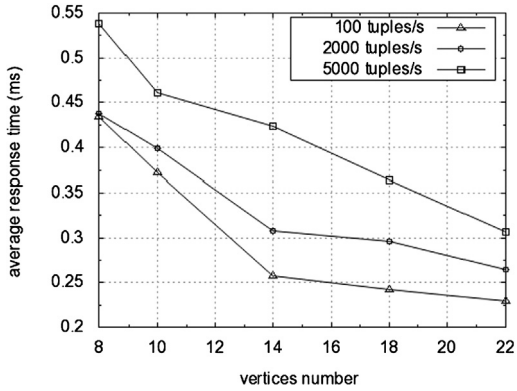


Fig. 10. Response time of DAG with different vertices instances.

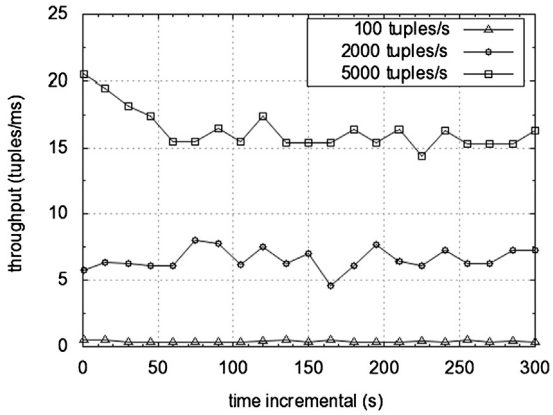


Fig. 11. System throughput of DAG different data stream rates.

5000 tuples/s, and the number of vertex instances is 22, the average response time is 0.229 ms, 0.264 ms, and 0.306 ms, respectively.

(2) *System throughput*. System throughput ST reflects the data rate of a data center when processing the same data stream by a DAG. ST can be described by tuples per each millisecond of all the applications in a big data stream computing environment.

When the rate of a data stream is stable, with the increase of time, the system throughput of DAG has some fluctuations. When the time is long enough, the system throughput will stabilize at a reasonable level. As shown in Fig. 11, when the rate of the data stream is stable at 100 tuples/s, 2000 tuples/s, and 5000 tuples/s, the system throughput is 0.367 tuples/ms, 6.609 tuples/ms, and 16.329 tuples/ms, respectively.

When the rate of a data stream is stable, with the increase of the number of vertex instances, the average system throughput of DAG will decrease. As shown in Fig. 12, when the rate of the data stream is stable at 100 tuples/s, 2000 tuples/s, and 5000 tuples/s, and the number of vertex instances is 22, the average system throughput is 0.463 tuples/ms, 11.491 tuples/ms, and 24.991 tuples/ms, respectively.

(3) *System reliability*. System reliability $R(sys)$ reflects the total reliability of a data center when processing the same data stream by a DAG. $R(sys)$ can be calculated by the reliability of all applications in a big data stream computing environment, and can be obtained by (40).

$$R(sys) = \prod_{i=1}^{n} r(G_i). \tag{40}$$

When the rate of a data stream is stable at 100 tuples/s, with the increase of the vertex number, the FTDG allocation algorithm can improve the system reliability to a degree, the more the number of vertices, the greater degree of the improvement in system reliability. As shown in Fig. 13, comparing to the basic Round-Robin allocation strategy in Storm, and the improved allocation strategy in JStorm [20], which is built on Storm, it is a great improvement in system reliability in the data center.

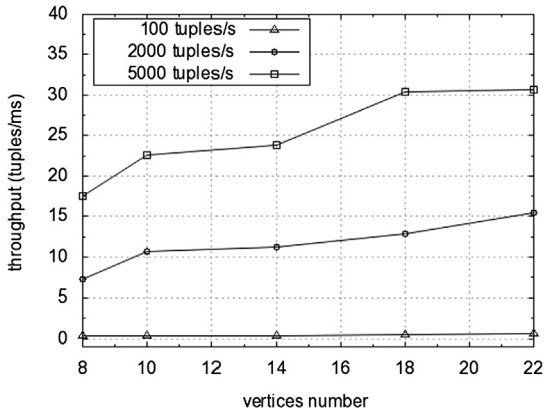


Fig. 12. System throughput of DAG with different vertex instances.

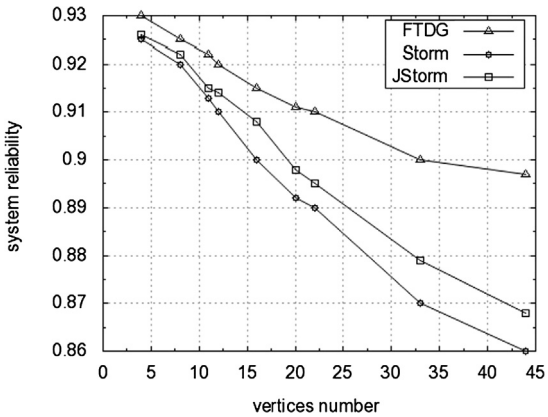


Fig. 13. System reliability.

7. Related work

In this section, three broad categories of related work are presented: big data stream computing, performance-oriented data stream scheduling, and fault tolerance strategy in distributed heterogeneous computing systems.

7.1. Big data stream computing

Big data stream computing is a new trend for future computing with the quantity of data growing. In general, there are two main mechanisms for big data computing, big data stream computing and big data batch computing. Big data stream computing is the model of straight through computing, such as Storm [20] S4 [30], while big data batch computing is the model of storing then computing, such as MapReduce framework [31], open sourced by the Hadoop implementation [32]. However, big data batch computing is not adequate for supporting big data stream computing.

In [11], a distributed-system infrastructure, TimeStream, is designed specifically for reliable low-latency continuous computing of big streaming data on large clusters of commodity machines. The big streaming data has the following characteristics: (1) High volumes, the incoming data arrives continuously at volumes that far exceeds the capabilities of individual machines; (2) Low latency, input streams incur multi-staged computing at low latency to produce output streams. A powerful abstraction, called resilient substitution, serves as a uniform foundation for handling failure recovery and dynamic reconfiguration correctly and efficiently.

In [33], a distributed system, Naiad, is designed specifically for executing data parallel, cyclic dataflow programs. It offers the high throughput of batch computing, the low latency of stream computing, and the ability to perform iterative and incremental computing. A computational model, timely dataflow, enriches dataflow computation with timestamps that represent logical points in the computation and provide the basis for an efficient, lightweight coordination mechanism. Many powerful high-level programming models can be built on Naiad's low-level primitives, enabling such diverse tasks as streaming data analysis, iterative machine learning, and interactive graph mining.

In [34], a distributed computing model, discretized streams (D-Streams), is implemented in a system named Spark Streaming. D-Streams is designed specifically for fast, often sub-second, recovery from faults and stragglers, without the

overhead of replication, in ever-larger scales of data stream computing environments. D-Streams enable a parallel recovery mechanism that improves efficiency over traditional replication and backup schemes, and tolerates stragglers. The idea in D-Streams is to structure a streaming computation as a series of stateless, deterministic batch computations in small time intervals. D-Streams make (1) the state at each timestep fully deterministic given the input data, forgoing the need for synchronization protocols, and (2) the dependencies between this state and older data visible at affine granularity.

To summarize, current big data computing schemes are not limited in one or other aspect. Up to now, most of the research efforts are in big data batch computing, while the response time of batch computing, at the level of several seconds or even minutes, cannot meet the request of most big data applications. However, the research about big data stream computing is not enough and not systematic, such as, Storm system employ a round-robin strategy as its task scheduling strategy, which is always not efficient and effective. So it needs to pay much attention to stream computing in big data and cloud era.

7.2. Performance-oriented data stream scheduling

In distributed stream computing environments, the structure of applications of data streams is usually described as DAG. Performance oriented data stream scheduling tries to dynamically schedule recourses for many DAGs according to dynamic changes in the volume of the data stream, which is known to be NP-hard. The performance oriented data stream scheduling can be further classified into the follow groups [35]: list scheduling algorithms, task duplication-based scheduling algorithms, clustering scheduling algorithms, and guided random search scheduling algorithms.

In [36], a two-phase list-based scheduling algorithm, named Hybrid Heuristic-Genetic Scheduling (H2GS) algorithm, for data stream scheduling on heterogeneous distributed computing systems, is proposed. The first phase implements a heuristic list-based algorithm, named LDCP, to generate a high quality schedule. In the second phase, the LDCP-generated schedule is injected into the initial population of a customized genetic algorithm, which proceeds to evolve shorter schedules.

Task duplication is a well-known technique for reducing the necessary communication between the processors. Some certain crucial tasks will be duplicated and executed on more than one processor. In [37], a contention-aware task duplication scheduling algorithm is proposed. It works under the general contention model, and its algorithmic components are based on state-of-the-art techniques used in task duplication and contention-aware algorithms.

Data streams are infinite and evolving over time, and do not have any knowledge about the number of clusters. Clustering data streams will help overcome the various factors and noise in data streams. In [38], density-based data stream clustering algorithms are surveyed. In [39], a data stream clustering algorithm, named SVStream, is proposed. SVStream is based on support vector domain description and support vector clustering. The data elements of a stream are mapped into a kernel space, and the support vectors are used as the summary information of the historical elements to construct cluster boundaries of arbitrary shape. To adapt to both dramatic and gradual changes, multiple spheres are dynamically maintained, each describing the corresponding data domain presented in the data stream.

In [40], a Double Molecular Structure-based Chemical Reaction Optimization (DMSCRO) algorithm, for Directed Acyclic Group data stream scheduling on heterogeneous computing systems, is developed. In DMSCRO, one molecular structure is used to encode the execution order of the tasks in a DAG job, while the other molecular structure to encode the task-to-computing-node mapping. DMSCRO also designs the necessary elementary chemical reaction operations and the fitness function suitable for the scenario of DAG scheduling.

To summarize, current performance oriented data stream scheduling is not limited in one or other aspect. Up to now, most of the researches are in static scheduling, all the information about scheduling is estimate and unchanged, and must be known in advance. However, when the volume of the data stream has changed, the scheduling may not be a wise strategy. If this static scheduling is employed again in this stage, huge fluctuations will occur. In big data stream environments, the volume of data stream is always changing.

7.3. Fault tolerance strategy in distributed heterogeneous computing system

In distributed heterogeneous computing, fault tolerance strategy has been widely studied. High system reliability has become a key metric for evaluating how good a computing system is. Recently, more and more research works began to address the DAG scheduling problem with fault tolerance issue in mind.

In [8], an application reliability analysis model based on Weibull distribution is built, and a reliability-driven earliest finish time with duplication scheduling algorithm REFTD is proposed. REFTD incorporates task reliability overhead into scheduling. In order to improve system reliability, it duplicates task as if task hazard rate is more than a threshold, and it can also reduce the scheduling length of applications in a heterogeneous computing system with arbitrary connected networks.

In [9], a distributed scientific workflow scheduling strategy, named dis-DRMED, for maximized reliability under certain end-to-end delay bound, is proposed. dis-DRMED considers both the maximum reliability and the minimum end-to-end delay in a two-step procedure. In the first step, a scheduling algorithm combining iterative Critical Path search and Layer-based priority assigning techniques (CPL) is adopted to minimize the end-to-end delay by focusing on the optimal allocation of tasks on the critical path. In the second step, tasks on noncritical paths are rescheduled to improve the overall execution reliability.

In [41], a framework for adaptive fault-tolerant workflow execution in the grid environments is proposed. The framework can dynamically decide the most appropriate fault tolerance technique with a user-defined rule-based system. The main work is done in twofold: the developed framework and the model-based dependability analysis. The purpose in carrying out a model-based dependability analysis consists of evaluating the interaction between the framework and the distributed grid environment beyond the physical limitations of an empirical evaluation.

In [42], provisioning and job reconfiguration techniques are proposed for adapting to execution environment changes when processing data streams on cluster-based deployments. This work targets data intensive applications where the inter-node transfer latency is significant, and aims to minimize the transfer latency. These techniques are based on a general group-based job representation that is commonly found in many distributed data stream processing frameworks.

Existing reliability aware measurement and management mechanism proposed for distributed stream computing systems cannot be directly implemented for big data stream computing. As they just base on special assumptions, focus on maximizing system reliability, or try to balance energy and performance. In contrast, our solution can achieve high fault tolerance without affecting the minimum response time. All vertices in DAG will be divided into two groups, vertices on all critical path and vertices on non-critical paths. Low response time is only determined by all the vertices on the critical path. Furthermore, maximizing the system reliability can be achieved by consolidating recourse for vertices on non-critical path. Our solution intends to achieve high fault tolerance and low response time objectives measurement and management in big data stream computing environments.

8. Conclusions and future directions

In big data stream computing environments, the data centers are usually composed of tens of thousands of diverse sets of computing nodes with different capabilities, and are interconnected with arbitrary network architectures. Failures are inevitable due to the inherently unreliable nature of the computing nodes and communication links. It is needed to consider fault tolerance in data centers, while maintaining high performance and service level objectives. To achieve reliable stream computing in big data computing environments and to address high fault tolerance by carefully choosing the running computing nodes and communication links in data centers, it is important to obtain a clear picture of the total reliability of the computing nodes and communication links in big data environments. More importantly, it is needed to understand how to maximize the system reliability and minimize the response time in data centers, and to deal with the trade-off between high fault tolerance and low response time objectives efficiently and effectively.

In this paper, we have covered all the four aspects of FTDG.

(1) Identifying the critical path of a data stream graph at a given data stream throughput, and quantifying the system reliability of a data stream graph.

(2) Allocating tasks by the fault tolerance aware heuristic and critical path scheduling mechanism.

(3) Optimizing the task scheduling by reallocating the critical vertices on the critical path of the data stream graph to lower the response time and reducing system fluctuations.

(4) Prototype implementation, simulation, and performance evaluation of the proposed FTDG.

In the future, we will focus on the research as follows:

(1) Optimizing the structure of the data stream graph, and providing an efficient data stream graph for each application.

(2) Deploying the FTDG on a real big data stream computing platform.

Acknowledgments

This work was supported by the National Fundamental Research Program of China 973 Program under Grant No. 2014CB340402; the National Natural Science Foundation of China under Grant No. 61272055, No. 61672315, No. 61502012, and No. 61602428; and the Fundamental Research Funds for the Central Universities under Grant No. 2652015338.

References

[1] M.M. Rathore, A. Ahmad, A. Paul, Real time intrusion detection system for ultra-high-speed big data environments, J. Supercomput. 72 (9) (Sep. 2016) 3489–3510.

[2] A. Siddiqa, I.A.T. Hashem, I. Yaqoob, M. Marjani, S. Shamshirband, A. Gani, F. Nasaruddin, A survey of big data management: taxonomy and state-of-the-art, J. Netw. Comput. Appl. 71 (Aug. 2016) 151–166.

[3] J. Lu, D. Li, Bias correction in a small sample from big data, IEEE Trans. Knowl. Data Eng. 25 (11) (Nov. 2013) 2658–2663.

[4] J.M. Tien, Big data: unleashing information, J. Syst. Sci. Syst. Eng. 22 (2) (Jun. 2013) 127–151.

[5] C. Hu, Z. Xu, Y. Liu, L. Mei, L. Chen, X. Luo, Semantic link network-based model for organizing multimedia big data, IEEE Trans. Emerg. Top. Comput. 2 (3) (Sept. 2014) 376–387.

[6] M. Andreolini, M. Colajanni, M. Pietri, S. Tosi, Adaptive, scalable and reliable monitoring of big data on clouds, J. Parallel Distrib. Comput. 79–80 (May 2015) 67–79.

[7] Z. Xu, Y.H. Liu, L. Mei, C.P. Hu, L. Chen, Semantic based representing and organizing surveillance big data using video structural description technology, J. Syst. Softw. 102 (Apr. 2015) 217–225.

[8] X.Y. Tang, K.L. Li, G.P. Liao, An effective reliability-driven technique of allocating tasks on heterogeneous cluster systems, Clust. Comput. 17 (4) (Dec. 2014) 1413–1425.

[9] F. Cao, M.M. Zhu, Distributed workflow mapping algorithm for maximized reliability under end-to-end delay constraint, J. Supercomput. 66 (3) (Dec. 2013) 1462–1488.
[10] J. Xu, Z. Chen, J. Tang, S. Su, T-storm: traffic-aware online scheduling in storm, in: Proc. 2014 IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014, IEEE Press, Aug. 2014, pp. 535–544.
[11] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, Z. Zhang, TimeStream: reliable stream computation in the cloud, in: Proc. 8th ACM European Conference on Computer Systems, EuroSys 2013, ACM Press, Apr. 2013, pp. 1–14.
[12] M. Sharifi, S. Shahrivari, H. Salimi, PASTA: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources, Computing 95 (1) (Jan. 2013) 67–88.
[13] J. Xuan, X. Luo, G. Zhang, J. Lu, Z. Xu, Uncertainty analysis for the keyword system of web events, IEEE Trans. Syst. Man Cybern. Syst. 46 (6) (Jun. 2016) 829–842.
[14] Z. Xu, X. Wei, X.F. Luo, Y.H. Liu, L. Mei, C.P. Hu, L. Chen, Knowle: a semantic link network based system for organizing large scale online news events, Future Gener. Comput. Syst. 43–44 (Feb. 2015) 40–50.
[15] D. Berberidis, V. Kekatos, G.B. Giannakis, Online censoring for large-scale regressions with application to streaming big data, IEEE Trans. Signal Process. 64 (15) (Aug. 2016) 3854–3867.
[16] R.C. Fernandez, M. Migliavacca, E. Kalyvianaki, P. Pietzuch, Integrating scale out and fault tolerance in stream processing using operator state management, in: Proc. 2013 ACM SIGMOD Conference on Management of Data, SIGMOD 2013, ACM Press, Jun. 2013, pp. 725–736.
[17] L. Gu, D.Z. Zeng, S. Guo, Y. Xiang, J.K. Hu, A general communication cost optimization framework for big data stream processing in geo-distributed data centers, IEEE Trans. Comput. 65 (1) (Jan. 2016) 19–29.
[18] Z. Xu, Y. Liu, N.Y. Yen, L. Mei, X. Luo, X. Wei, C. Hu, Crowdsourcing based description of urban emergency events using social media big data, IEEE Trans. Cloud Comput. (2016) 1–11, http://dx.doi.org/10.1109/TCC.2016.2517638.
[19] S. Ren, N. Deligiannis, Y. Andreopoulos, M. Islam, D.S.M. Van, Dynamic scheduling for energy minimization in delay-sensitive stream mining, IEEE Trans. Signal Process. 62 (20) (Oct. 2014) 5439–5448.
[20] Storm, http://storm-project.net/.
[21] S. Schneider, M. Hirzel, B. Gedik, K.L. Wu, Safe data parallelism for general streaming, IEEE Trans. Comput. 64 (2) (Feb. 2015) 504–517.
[22] L. Zhao, Y. Ren, K. Sakurai, Reliable workflow scheduling with less resource redundancy, Parallel Comput. 39 (10) (Oct. 2013) 567–585.
[23] D. Sun, G. Zhang, S. Yang, W. Zheng, S.U. Khanb, K. Li, Re-Stream: real-time and energy-efficient resource scheduling in big data stream computing environments, Inf. Sci. 319 (Oct. 2015) 92–112.
[24] R.N. Calheiros, R. Buyya, Meeting deadlines of scientific workflows in public clouds with tasks replication, IEEE Trans. Parallel Distrib. Syst. 25 (7) (Jul. 2014) 1787–1796.
[25] X.F. Luo, Z. Xu, J. Yu, X. Chen, Building association link network for semantic link on web resources, IEEE Trans. Autom. Sci. Eng. 8 (3) (Jul. 2011) 482–494.
[26] J.E. Pezoa, M.M. Hayat, Performance and reliability of non-markovian heterogeneous distributed computing systems, IEEE Trans. Parallel Distrib. Syst. 23 (7) (Jul. 2012) 1288–1301.
[27] L.X. Zhang, K.L. Li, Y.M. Xu, J. Mei, F. Zhang, K.Q. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Inf. Sci. 319 (Oct. 2015) 113–131.
[28] M.M. Rathore, A. Ahmad, A. Paul, S. Rho, Urban planning and building smart cities based on the Internet of Things using Big Data analytics, Comput. Netw. 101 (Jun. 2016) 63–80.
[29] L. Aniello, R. Baldoni, L. Querzoni, Adaptive online scheduling in storm, in: Proc. the 7th ACM International Conference on Distributed Event-Based Systems, DEBS 2013, ACM Press, Jul. 2013, pp. 207–218.
[30] L. Neumeyer, B. Robbins, A. Nair, A. Kesari, S4: distributed stream computing platform, in: Proc. 10th IEEE International Conference on Data Mining Workshops, ICDMW 2010, IEEE Press, Dec. 2010, pp. 170–177.
[31] Y. Zhao, J. Wu, Dache: a data aware caching for big-data applications using the MapReduce framework, in: Proc. 32nd IEEE Conference on Computer Communications, INFOCOM 2013, IEEE Press, Apr. 2013, pp. 35–39.
[32] W. Shang, Z.M. Jiang, H. Hemmati, B. Adams, A.E. Hassan, P. Martin, Assisting developers of big data analytics applications when deploying on Hadoop clouds, in: Proc. 35th International Conference on Software Engineering, ICSE 2013, IEEE Press, May 2013, pp. 402–411.
[33] D.G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, M. Abadi, Naiad: a timely dataflow system, in: Proc. 24th ACM Symposium on Operating Systems Principles, SOSP 2013, ACM Press, Nov. 2013, pp. 439–455.
[34] M. Zaharia, T. Das, H.Y. Li, T. Hunter, S. Shenker, I. Stoica, Discretized streams: fault-tolerant streaming computation at scale, in: Proc. 24th ACM Symposium on Operating Systems Principles, SOSP 2013, ACM Press, Nov. 2013, pp. 423–438.
[35] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, J. Wang, Cost-efficient task scheduling for executing large programs in the cloud, Parallel Comput. 39 (4–5) (Apr.-May. 2013) 177–188.
[36] M.I. Daoud, N. Kharma, A hybrid heuristic-genetic algorithm for task scheduling in heterogeneous processor networks, J. Parallel Distrib. Comput. 71 (11) (Nov. 2011) 1518–1531.
[37] O. Sinnen, A. To, M. Kaur, Contention-aware scheduling with task duplication, J. Parallel Distrib. Comput. 71 (1) (Jan. 2011) 77–86.
[38] A. Amini, T.Y. Wah, H. Saboohi, On density-based data streams clustering algorithms: a survey, J. Comput. Sci. Technol. 29 (1) (Jan. 2014) 116–141.
[39] C.D. Wang, J.H. Lai, D. Huang, W.S. Zheng, SVStream: a support vector-based algorithm for clustering data streams, IEEE Trans. Knowl. Data Eng. 25 (6) (Jun. 2013) 1410–1424.
[40] Y. Xu, K. Li, L. He, T.K. Truong, A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization, J. Parallel Distrib. Comput. 73 (9) (Sept. 2013) 1306–1322.
[41] F.P. Guimaraes, P. Celestin, D.M. Batista, G.N. Rodrigues, A.C.M.A. de Melo, A framework for adaptive fault-tolerant execution of workflows in the grid: empirical and theoretical analysis, J. Grid Comput. 12 (1) (Mar. 2014) 127–151.
[42] A. Chatzistergiou, S.D. Viglas, Fast heuristics for near-optimal task allocation in data stream processing over clusters, in: Proc. the 23rd ACM International Conference on Conference on Information and Knowledge Management, ACM Press, Nov. 2014, pp. 1579–1588.