Information Sciences 319 (2015) 92–112

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments

Dawei Sun [a], Guangyan Zhang [a,*], Songlin Yang [a], Weimin Zheng [a], Samee U. Khan [b], Keqin Li [c]

[a] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[b] Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050, USA
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

A R T I C L E I N F O

Article history:
Received 14 June 2014
Received in revised form 17 February 2015
Accepted 10 March 2015
Available online 17 March 2015

Keywords:
Big data
Critical path
Directed acyclic graph
Energy efficiency
Real time
Stream computing

A B S T R A C T

To achieve high energy efficiency and low response time in big data stream computing environments, it is required to model an energy-efficient resource scheduling and optimization framework. In this paper, we propose a real-time and energy-efficient resource scheduling and optimization framework, termed the Re-Stream. Firstly, the Re-Stream profiles a mathematical relationship among energy consumption, response time, and resource utilization, and obtains the conditions to meet high energy efficiency and low response time. Secondly, a data stream graph is modeled by using the distributed stream computing theories, which identifies the critical path within the data stream graph. Such a methodology aids in calculating the energy consumption of a resource allocation scheme for a data stream graph at a given data stream speed. Thirdly, the Re-Stream allocates tasks by utilizing an energy-efficient heuristic and a critical path scheduling mechanism subject to the architectural requirements. This is done to optimize the scheduling mechanism online by reallocating the critical vertices on the critical path of a data stream graph to minimize the response time and system fluctuations. Moreover, the Re-Stream consolidates the non-critical vertices on the non-critical path so as to improve energy efficiency. We evaluate the Re-Stream to measure energy efficiency and response time for big data stream computing environments. The experimental results demonstrate that the Re-Stream has the ability to improve energy efficiency of a big data stream computing system, and to reduce average response time. The Re-Stream provides an elegant trade-off between increased energy efficiency and decreased response time effectively within big data stream computing environments.

1. Introduction

1.1. Background and motivation

Big data, which is too large, changing too fast, and too dispersed, is beyond the ability of available hardware and software facilities to undertake their acquisition, access, analytics, and/or application in a reasonable amount of time and space. Big data stream computing relies on programs that compute continuous data streams. Providing fast big data processing with large-scale parallel architectures, stream computing is becoming the fastest and most efficient solution to obtain useful

* Corresponding author.
E-mail addresses: sundaweicn@163.com (D. Sun), gyzh@tsinghua.edu.cn (G. Zhang), yangsl14@mails.tsinghua.edu.cn (S. Yang), zwm-dcs@tsinghua.edu.cn (W. Zheng), samee.khan@ndsu.edu (S.U. Khan), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.ins.2015.03.027
0020-0255/© 2015 Elsevier Inc. All rights reserved.

knowledge from big data. This allows organizations to react quickly when problems appear or to predict new trends [9,17,33,42]. Big data stream characterizes high-speed, real-time, and large-volumes of input data from applications. The data can be structured, unstructured, or of both types. Data sources of all such applications often take the form of continuous data streams. Moreover, many data streams from different data sources may form a mixture of the data types that may be incompatible. During the data stream processing, the data fusion phase is very complex, and is completely different from the process of the composition of the massive data streams [8,27,42]. In the current stream computing environments, high performance and service level objectives are considered as the main issues, while energy efficiency is ignored.

To achieve energy-efficient stream computing, it is important to obtain a clear picture of the total energy consumption within big data environments. However, the current state of the art does not directly address the complex tradeoff between energy consumption and response time [3,5,8,37]. This justifies the importance of modeling an energy-efficient resource scheduling and optimization framework for 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, while highly efficient resource scheduling strategy is ignored. For example, the instance number of task topology in the Storm system is statically set by users, cannot adapts to the change of data stream dynamically. The round-robin strategy is employed to allocate resources when a task topology is submitted to the system. Obviously, a resource scheduling strategy with the feature of elasticity and adaptively to changes of data stream is in need. The authors of [37] also tried to solve this problem. However, their solution only focuses on response time, while many factors are not considered, such as initial scheduling, critical path of task topology.

(2) The factor of energy efficiency is not considered.
At present, high energy efficiency is not considered in designing a big data stream computing system. For example, in the Storm system, some resources may be in low utility for a long time, while the system is in the high power state. It may not waste too much energy for a small-scale system. However, it may waste too much energy in big data stream computing environments, which are usually composed of several thousand computing nodes. Energy consumption is linearly associated with resource utilization. Resource consolidation is a major way for energy saving. Some energy saving technologies, such as resources consolidation, dynamic voltage and frequency scaling technology, should be considered in big data stream computing systems. In current big data stream computing systems, such optimization technologies are not considered.

So we try to achieve high energy efficiency and low response time in a big data stream computing environment. Our approach is partially inspired by the following ideas.

(1) Dynamic Voltage and Frequency Scaling (DVFS).
DVFS has been used to scale up or scale down the input voltage and operational frequency of processors. It is an effective way to reduce energy consumption. The lower the power state is, the greater the energy savings is [25,26,41]. However, the execution time also becomes longer. As shown in Fig. 1(a), when the task runs in a higher power state, the execution time is shorter and the task is finished in advance of deadline. When the task is finished, the machine works in an idle state, some energy is also required. As shown in Fig. 1(b), when the task runs in a lower power state, the execution time is longer but without causing the task to go beyond the deadline. By finishing task just in time, less energy is needed.

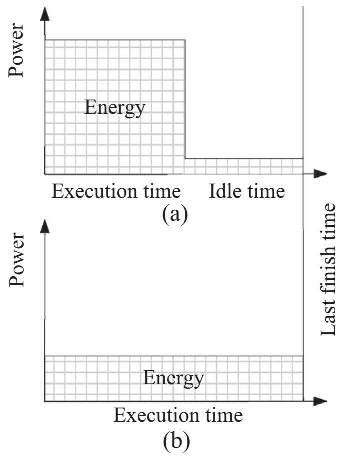

Fig. 1. Dynamic voltage and frequency scaling.

(2) Critical path in a directed acyclic graph (DAG).
In a DAG, the response time is determined by the latency of critical path in the DAG. As shown in Fig. 2, in a real-time stream computing environment, rescheduling all vertices to improve the response time is not a wise choice. That 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 critical path (10–21–25–12) of the DAG, while vertex 35 is not on the critical path. Vertex 25 is on critical path (10–21–25–12) of the DAG. If we reschedule vertex 25, the computing time is reduced from 25 ms to 20 ms. Accordingly, the response time of the DAG will also be reduced by 5 ms. However, if the computing time of vertex 25 is reduced form 25 ms to 15 ms, the response time of the DAG will be reduced by 6 ms instead of by 10 ms. This is because critical path varies and new critical path (10–35–5–12) will determine response time of the DAG.

All those observations can be used to optimize the real-time and energy-efficient scheduling framework. Motivated by aforementioned observations, in this paper, we design all four aspects of Re-Stream by conducting the following investigation.

(1) Profiling the mathematical relationship between energy consumption, response time, and resource utilization in big data stream computing, obtaining the conditions to meet the high energy efficiency, and low response time objectives.
(2) Modeling data stream graph in big data stream computing environments by referring to the distributed stream computing theories, identifying the critical path in a data stream graph, and getting energy consumption of an allocation scheme for a data stream graph at a special data stream speed.
(3) Allocating tasks by the energy efficient heuristic and the critical path scheduling mechanism subject to the constraints of tasks' architectural requirements, and optimizing the scheduling mechanism online by reallocating the critical vertices on critical path of data stream graph to minimize response time and system fluctuations, and consolidating non-critical vertices on non-critical path to maximize the energy efficiency.
(4) Implementing our approach, and evaluating it in big data stream computing environments for both energy efficiency and response time.

1.2. Contributions

Our contributions are summarized as follows:

(1) Formal definitions of data stream graph, big data stream computing architecture, and high energy efficiency and low response time in quantitative perspective, and systematical profiling of the mathematical relationship between energy consumption, response time, and resource utilization in big data stream computing environments.
(2) Systematic investigation in computation of nodes, communication among nodes, and allocating tasks by energy efficient heuristic and critical path scheduling mechanism subject to the constraints of tasks' architectural requirements.
(3) Identification of the critical path of a data stream graph at a given data stream throughput, and optimization of the scheduling mechanism online by reallocating the critical vertices on critical path of data stream graph to minimize response time and system fluctuations.
(4) Quantification of energy consumption of an allocation scheme for a data stream graph at a given data stream speed, and consolidation of non-critical vertices on non-critical path to maximize the energy efficiency.
(5) Prototype implementation, simulation, and performance evaluation of the proposed Re-Stream, which can provide a trade-off between high energy efficiency and low response time objectives efficiently and effectively in big data stream computing environments.

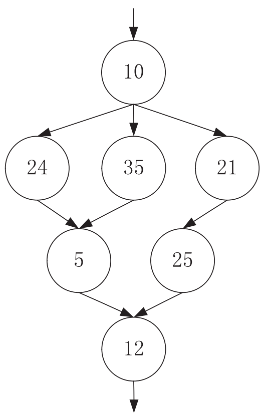

Fig. 2. Directed acyclic graph.

1.3. Paper organization

The remainder of this paper is organized as follows. In Section 2, a series of definitions of data stream graph and data stream graph scheduling model in big data stream computing environments are presented. Section 3 presents system architecture, workflow, critical vertex model, and energy consumption model of the Re-Stream. Section 4 focuses on the scheduling algorithm of the Re-Stream in a real big data stream computing platform. Section 5 shows the experimental environment, parameters setup, and experimental results of the Re-Stream. Section 6 reviews the related works on big data stream computing, performance aware data stream scheduling, and energy aware data stream scheduling. Finally, conclusions and future work are given in Section 7.

2. Problem statement

In big data stream computing environments, the stream application is partitioned into a set of sub-task wherein each sub-task depends on the results of executions of other sub-tasks, and is described into Data Stream Graph (DSG), which is described by directed acyclic graph (DAG). All those DSGs will be submitted to data center. To precisely reflect these factors in scheduling, optimizing, and evaluating DSG for big data stream computing, some related definitions are defined below [8,14,15,22,26,35].

2.1. Data stream graph

A stream is a sequence of data sets. A continuous stream is an infinite sequence of data sets, and the parallel streams are more than one stream to be processed at the same time. A stream computing is a program that computes continuous data streams. In stream computing, the multiple continuous parallel data streams can be processed by DSG, and is described by DAG. A measurable DSG view can be defined by Definition 1.

Definition 1 (*DSG*). A DSG G is a directed acyclic graph, composed of vertices set and directed edges set, has a logical structure and special function, and denoted as $G = (V(G), E(G))$, where $V(G) = \{v_1, v_2, \ldots, v_n\}$ is a finite set of n vertices, that represent sub-tasks, and $E(G) = \{e_{1,2}, e_{1,3}, \ldots, e_{n-i,n}\}$ is a finite set of directed edges, that represent 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 a ordered pair, where data stream come from v_i, and go to v_j.

The vertex v_i is characterized by a five-tuple $v_i = \left(id_{v_i}, f_{v_i}, c_{v_i}, i_{v_i}, o_{v_i}\right)$, 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, o_{v_i} is the output data stream for vertex v_i.

The vertex v_i represents a task composed of a set of instructions, which 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 DSG G is achieved by all n vertices, that is $O = F(I)$, where O, F, and I are output data steam, function of DSG G, and input data steam, respectively.

The directed edge $e_{i,j}$ is the directed communication link from vertex v_i to v_j, and is characterized by a tuple, that is $e_{i,j} = \left(id_{e_{i,j}}, c_{e_{i,j}}\right)$, where $id_{e_{i,j}}$ is the identification of directed edge $e_{i,j}$, $c_{e_{i,j}}$ 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. The source vertex is the vertex whose in-degree is zero, and the end vertex is the vertex whose out-degree is zero. A DSG G has at least one source vertex and one end vertex.

The scenario of a DSG with eleven vertices is shown in Fig. 3, where the vertices set $V = \{v_a, v_b, \ldots, v_k\}$, the directed edges $E = \{e_{a,c}, e_{b,c}, \ldots, e_{j,k}\}$, the source vertices are v_a and v_b, and the end vertex is v_k, the in-degree of vertex v_d is one, the out-degree of vertex v_d is two.

DSG G can be further divide into two groups, data stream logical graph and data stream instance graph. The data stream logical graph reflects the logical structure of a data steam graph G, and each vertex of G only has one instance. However, the data stream instance graph is actually deployed graph on a set of computing nodes, and some vertex may have many instances to eliminate the bottleneck and split the data stream. As shown in Fig. 4, the Storm source code is the main part to achieve TOP_N computing function, the corresponded logical graph is shown in Fig. 5(a), and the instance graph is shown in Fig. 5(b), where vertex v_a is parallelized into four instances, vertex v_b and vertex v_c are parallelized into three instances. In our work, we optimize the logical and instance graphs. As the DSG G can be allocated to computing nodes for execution and dynamically reconfigured the number of vertex at runtime.

Definition 2 (*sub-graph*). A sub-graph *sub-G* of the DSG 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 DSG G, v must also be in the *sub-G* if v is on a directed path from v_i to v_j, that is $\forall v_i, v_j \in V(sub - G)$, $\forall v \in V(G)$, if $v \in V(p(v_i, v_j))$, then $v \in V(p(sub - \mathrm{G}))$.

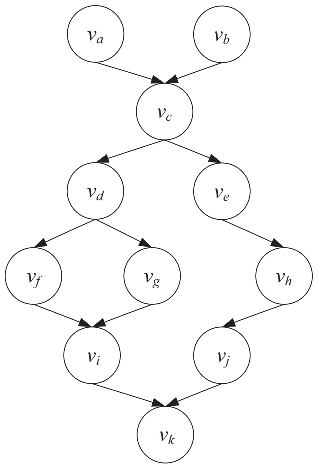

Fig. 3. A DSG.

```
builder.setSpout("a", new TestWordSpout(), 4);
builder.setBolt("b", new RollingCountBolt(9, 3), 3).
    fieldsGrouping("a", new Fields("word"));
builder.setBolt("c", new IntermediateRankingsBolt(TOP_N), 3).
    fieldsGrouping("b", new Fields("obj"));
builder.setBolt("d", new TotalRankingsBolt(TOP_N)).
    globalGrouping("c");
```

Fig. 4. Storm source code.

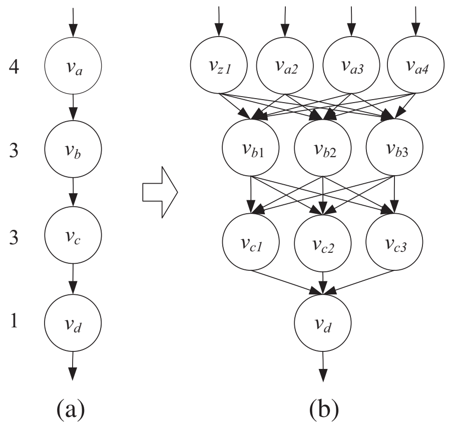

Fig. 5. (a) Logic graph, (b) instance graph.

A sub-graph *sub-G* is logically equivalent and can be substituted by a vertex. But reducing that sub-graph to a single logical vertex would create a graph with cycle, not a DAG.

The scenario of a DSG is shown in Fig. 3, which is an example of DSG consisting of eleven vertices, with v_a being a source vertex, and v_k being a result vertex. The sub-graph v_d, v_f, v_g and v_i, as well as all their edges, is a valid sub-graph, The sub-graph v_e, v_h and v_j also is a valid sub-graph, but the sub-graph v_d, v_f and v_i is not a valid sub-graph as v_g is on the path from v_d to v_i, but not part of the sub-graph.

Definition 3 (*Path*). A path $p(v_u, v_v)$ from vertex v_u to vertex v_v is a subset of $E(p(v_u, v_v))$ should meet conditions: $\exists k, e_{i,k} \in p(v_u, v_v),\ e_{k,j} \in p(v_u, v_v)$, for any directed edge $e_{k,l}$ in path $p(v_u, v_v)$ displays the properties: if $k \neq i$, then $\exists m$, and $e_{m,k} \in p(v_u, v_v)$; if $i \neq j$, then $\exists m$, and $e_{l,m} \in p(v_u, v_v)$.

The latencies $l_p(v_u, v_v)$ of a path from vertex v_u to vertex v_v is the sum of latencies of both vertices and edges on the path, as given by Eq. (1):

$$l_p(v_u, v_v) = \sum_{v_i \in V(p(v_u,v_v))} c_{v_i} | c_{V_{pree}} + \sum_{e_{i,j} \in E(p(v_u,v_v))} c_{e_{i,j}} | c_{E_{pree}}, c_{v_i}, c_{e_{i,j}} \geqslant 0. \tag{1}$$

If there are m paths from source vertex v_s to end vertex v_e in DSG G, then the latencies $l(G)$ of DSG G is given by Eq. (2):

$$l(G) = \max\left\{l_{p_1}(v_s, v_e), l_{p_2}(v_s, v_e), \ldots, l_{p_m}(v_s, v_e)\right\}. \tag{2}$$

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

According to (1),

$$l(G) = \max\left\{\begin{array}{l} \sum\limits_{v_i \in V(p_1(v_u,v_v))} c_{v_i} | c_{V_{pree}} + \sum\limits_{e_{i,j} \in E(p_1(v_u,v_v))} c_{e_{i,j}} | c_{E_{pree}}, \\ \sum\limits_{v_i \in V(p_2(v_u,v_v))} c_{v_i} | c_{V_{pree}} + \sum\limits_{e_{i,j} \in E(p_2(v_u,v_v))} c_{e_{i,j}} | c_{E_{pree}}, \\ \ldots, \\ \sum\limits_{v_i \in V(p_m(v_u,v_v))} c_{v_i} | c_{V_{pree}} + \sum\limits_{e_{i,j} \in E(p_m(v_u,v_v))} c_{e_{i,j}} | c_{E_{pree}} \end{array}\right\}. \tag{3}$$

In DSG G, if $\exists e_{i,j}$ from vertex v_i to vertex v_j, then vertex v_i is a parent of vertex v_j, and vertex v_j is a child of vertex v_i.

Definition 4 (*Topological Sort (TS)*). A $TS(G) = (v_{x_1}, v_{x_2}, \ldots, v_{x_n})$ of the vertices $V(G)$ in DSG G is a linear ordering of its vertices such that for every directed edge e_{x_i,x_j} ($e_{x_i,x_j} \in E(G)$) from vertex v_{x_i} to vertex v_{x_j}, v_{x_i} comes before v_{x_j} in the topological ordering.

A TS is possible if and only if the graph has no directed cycles, that is, it is need to be a DAG. Any DAG has at least one topological ordering.

In the scenario of a DSG shown in Fig. 3, $(v_b, v_a, v_c, v_d, v_e, v_h, v_f, v_g, v_i, v_j, v_k)$ is a valid TS.

Definition 5 (*Graph Partitioning (GP)*). A $GP(G) = \{GP_1, GP_2, \ldots, GP_m\}$ of the DSG G is a TS based split the vertex set $V(G)$ and correspond direct edges. A graph partitioning should meet non-overlapping and covering properties, that is if DSG is split into m sub graphs, $\forall i \neq j, i, j \in [1, m]$, then $GP_i \cap GP_j = \emptyset$, and $\bigcup_{i=1}^{m} GP_i = V(G)$.

In the scenario of a DSG shown in Fig. 3, $GP(G) = \{GP_1, GP_2, GP_3, GP_4, GP_5\}$, and $GP_1 = \{v_b, v_a, v_c\}$, $GP_2 = \{v_d, v_e\}$, $GP_3 = \{v_h, v_f, v_g\}$, $GP_4 = \{v_i, v_j\}$, $GP_5 = \{v_k\}$ is a valid graph partitioning. However, $GP(G) = \{GP_1, GP_2, GP_3, GP_4\}$, and $GP_1 = \{v_b, v_a, v_c\}$, $GP_2 = \{v_c, v_d, v_e\}$, $GP_3 = \{v_h, v_f, v_g\}$, $GP_4 = \{v_i, v_j\}$ is not a valid graph partitioning, as $GP_1 \cap GP_2 \neq \emptyset$, which does not meet the non-overlapping properties, and $\bigcup_{i=1}^{4} GP_i \neq V(G)$, and also does not meet the covering properties.

2.2. Data stream graph scheduling

A DSG scheduling system consists of a user broker, scheduling broker, and data centers. A scheduling broker is the data center central managing broker. Some specific features of a DSG scheduling system can be described as follows.

Let $U = \{u_1, u_2, \ldots, u_m\}$ be a user set composed of m users, $Gs = \{Gs_1, Gs_2, \ldots, Gs_m\}$ be a set of DSGs of the user set U, and $Gs_j = \left\{G_{j_1}, G_{j_1}, \ldots, G_{j_{m_j}}\right\}$ be a sub-set of DSGs of the jth user u_j, where m_j is the number of DSGs of the jth user u_j, and Gs_k is the kth DSG 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 data center. If u_0 has two DSGs, then $Gs_0 = \{G_{0_1}, G_{0_2}\}$, and $m_0 = 2$.

For simplicity, we assume that the DSG are non-interruptible, which means that as DSG starts its schedule, it cannot be interrupted until all vertices in this graph are completely allocated in data center.

Let $DC = \{cn_1, cn_2, \ldots, cn_n\}$ be a 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. The computation can be overlapped with communication. 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, and cp_k can vary from $cp_k^{\min}$ to $cp_k^{\max}$ with the help of Dynamic Voltage Scaling (DVS), where $0 < cp_k^{\min} < cp_k^{\max}$.

The scheduling probability matrix of vertices in DSG 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 that vertex v_j in a DSG is submitted to computing node cn_i, $p_{ij} \geqslant 0$, and $\sum_{i=0}^{n} p_{ij} = 1$. To keep computing node cn_i in the serviceability state,

cn_i should meet the constraint defined by (4), and the data center DC should meet the constraint which is defined by (5), so that tasks will not be generated exceed the rate at which tasks can be processed.

$$\sum_{j=0}^{m} p_{ij} \cdot cdr_j \leqslant cp_i \cdot (1 - cf_i). \tag{4}$$

$$\sum_{i=0}^{n}\sum_{j=0}^{m} p_{ij} \cdot cdr_j \leqslant \sum_{i=0}^{n} cp_i \cdot (1 - cf_i). \tag{5}$$

where cdr_j is the data stream arrive rate of vertex v_j on computing node cn_k.

The DSG scheduling model of allocate DSGs to computing nodes in data center can be defined by Definition 6.

Definition 6 (*DSG scheduling model*). Let the DSG scheduling model Gm of a big data stream computing system be represented by a four-tuple $Gm = (U, DC, Of, \Theta)$, where $U = \{u_1, u_2, \ldots, u_m\}$ is a user set composed of m users, and each user may request service independently with each other. $DC = \{cn_1, cn_2, \ldots, cn_n\}$ is a data center composed of n computing nodes, which are running in the form of virtual machines on physical machines. For each DSG, Of is an objective function for scheduling each DSG in big data computing environment, which is defined according to (6),

$$Of(l(G), en(G)) = \min(en(G)|l(G)) \ and \ \min(l(G)). \tag{6}$$

so as to maximize the energy efficiency and minimize response time, and achieved by the real-time energy-efficient resource scheduling and optimization framework Re-Stream, and Θ is an algorithm which achieves all those optimal strategies.

The $\min(l(G))$ can be achieved by minimum the critical path of DSG G, as critical path is the longest path of G. that is,

$$\min(l(G)) = \min\left(\max\left\{l_{p_1}(v_s, v_e), \ldots, l_{p_m}(v_s, v_e)\right\}\right). \tag{7}$$

The $\min(en(G))$ is the energy consumption of all computing nodes to run all those vertices in DSG G. It will be achieved by consolidating vertices running on non-critical path, and dynamically adjusting the voltage of the computing nodes running those vertices on non-critical path, with the prerequisite that those energy strategies do not affect the minimum the response time of DSG G.

$$\min(en(G)) = \sum_{\forall |v_i| \in V(G)} en(cn_j) \cdot x_{ij}, \tag{8}$$

subject to,

$$x_{ij} = \begin{cases} 1, if \ \ v_i \ running \ on \ cn_j, \\ 0, otherwise. \end{cases} \tag{9}$$

If one vertex v_i or many vertices running on a computing node cn_j, then only one $en(cn_j)$ is calculated in DSG G, and denoted as $|v_i|$.

3. Overview of the Re-Stream framework

In this section, we focus our attention on Re-Stream in a big data stream computing environment. The overview of the Re-Stream framework includes system architecture, workflow description, critical path model, and energy consumption model [10,12,13,18,21,23,30].

3.1. System architecture

The system architecture of Re-Stream is composed of four spaces. These are hardware space, Storm space, graph space, and user space as shown in Fig. 6.

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

In the Storm space, performance aware dynamic scheduling of critical vertices model and energy aware dynamic scheduling of non-critical vertices model are employed to support real-time energy-efficient resource scheduling and optimization objectives based on the open source Storm platform [31]. The performance aware dynamic scheduling of critical vertices model is used to minimize the response time of a special DSG. The energy aware dynamic scheduling of non-critical vertices model is used to minimize energy consumption while maintaining the corresponding response time.

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

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

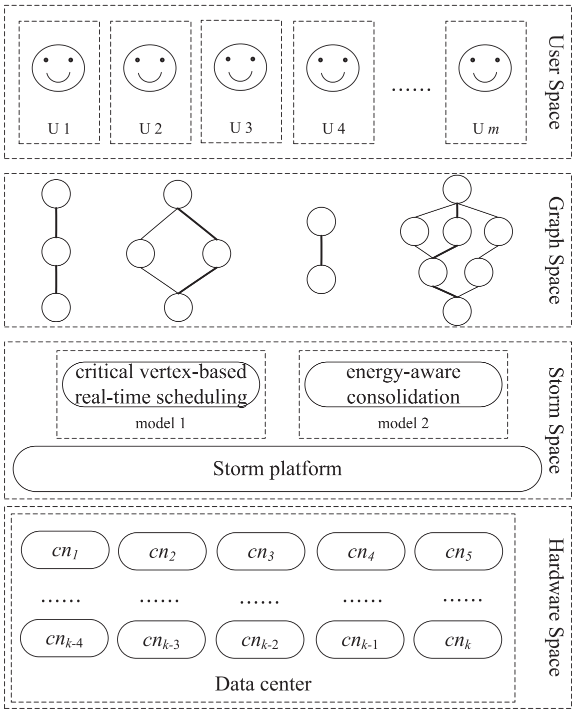

Fig. 6. Re-Stream system architecture.

Definition 7 (*DSG optimizing cycle*). A DSG optimizing cycle can be divided into four phases, profiling, quantifying, modeling, and evaluating as shown in Fig. 7. Periodically running through these four phases is able to establish a kind of in-time feedback, and to allow fast adaption to real-time changing big data stream computing environment.

In the profiling phase, it is desired to profile a mathematical relationship between energy consumption, response time, and resource utilization, and to obtain the conditions that meet the high energy efficiency and low response time.

In the modeling phase, it is desired to model DSG in big data stream computing environments by referring to the distributed stream computing theories that identifies the critical path in DSG, and getting energy consumption of an allocation scheme for a DSG at a special data stream speed.

In the scheduling phase, it is desired to schedule task by energy efficient heuristic critical path scheduling mechanism subject to the constraints of tasks' architectural requirements, and online optimization of the scheduling mechanism by performance aware dynamic scheduling of critical vertices on critical path of DSG to minimize response time and system fluctuations. It helps to consolidate non-critical vertices on non-critical path by energy aware dynamic scheduling to maximize the energy efficiency that meets high energy efficiency and low response times.

At last, in the evaluating phase, it is desired to evaluate the high energy efficiency and low response time, and to consider high energy efficiency and low response time.

3.2. Workflow description

To achieve high energy efficiency and low response time objectives, we employ Re-Stream based on the Storm platform. To maximize energy efficiency without affecting the minimized response time, all vertices in DSG are divided into two

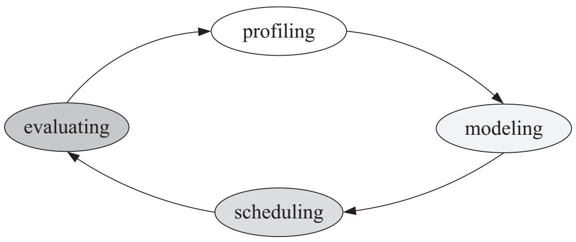

Fig. 7. DSG optimizing cycle.

groups, vertices on critical path, and vertices on non-critical path. Minimal response time is determined by vertices on critical path. Moreover, maximizing the energy efficiency can be achieved by consolidation recourse for vertices on non-critical path.

In the first phase, the computation cost of each vertex and the communication cost of each edge in each DSG are estimated in the currently data stream environments, the critical path and critical vertex can be calculated. When the DSG is first placement, a critical path based earliest completion time priority scheduling strategy is employed, in which computing cost and communication cost are under consideration. In this strategy, the vertex on critical path will be priority schedule to the earliest completion computing node only if its predecessors have been scheduled. If predecessors of this vertex have not been scheduled, then predecessors will be prior scheduled to the earliest completion computing node based on a topology sort. Hereafter, when arrival rate of data stream changes, the response time may also be changed, and then a critical vertex on critical path of a DSG is selected and scheduled to a new computing node to achieve minimum response time and system fluctuations. The energy consumed is estimated for all computing nodes.

In the second phase, an energy model is employed to estimate the energy consumption of each computing node in the currently data stream environments. An energy aware scheduling of non-critical vertices is employed to maximize the energy efficiency without increasing the response time of currently DSG. Usually, the more vertices of a DSG, the higher energy efficiency achieved along with, the lower CUP utilization of each computing node, and the higher energy efficiency.

The Re-Stream operates online in big data stream computing environments. To the best of our knowledge, no research have been done to tradeoff energy efficiency and response time for DSG in big data stream computing. The workflow of Re-Stream is shown in Fig. 8, which is a schematic for showing the workflow of Re-Stream. To show vertex instance running on different threads, processes, or computing nodes, it is intended to reflect the communication cost among threads, among process, and among computing nodes are not in the same levels, as usually, the communication cost among threads is the least, the communication cost among process is medium, the communication cost among computing nodes is the most. The resources and tasks on different computing nodes are different, so the computation cost of different computing nodes are different. The communication cost and computation cost are very important factors for Re-Stream. More details are presented in the remainder of this section. The algorithm implication of Re-Stream is presented in the next section.

3.3. Critical path model

In a big data stream computing environment, for a DSG 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, where each computation cost c_{v_i,cn_j} (is described by second) is the time required when running vertex v_i on

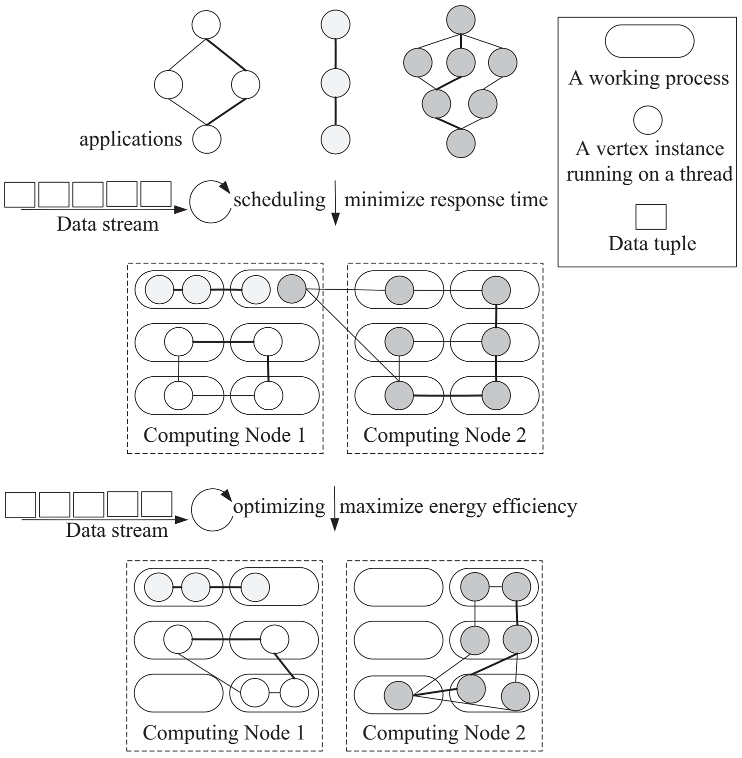

Fig. 8. Re-Stream workflow.

computing node cn_j, and is related with the time complex degree t_{v_i} (is described by million instructions) of the tasks in vertex v_i, the input data rate ir_{v_i} (is described by million bit per second) of vertex v_i in the system, and the processing ability p_{cn_j} (is described by million instructions per second) of the computing node cn_j, and can be described as (10):

$$c_{v_i,cn_j} = F_{c_v}\left(t_{v_i}, ir_{v_i}, p_{cn_j}\right). \tag{10}$$

For each data tuple, computation cost c_{v_i,cn_j} is calculated by (11):

$$c_{v_i,cn_j} = \frac{ir_{v_i}}{p_{cn_j}}. \tag{11}$$

The communication cost $c_{e_{i,j}}$ of directed edge $e_{i,j}$ is the time required when transmission data stream from vertex v_i to v_j. If v_i and v_j run on the same computing node, then $c_{e_{i,j}} = 0$. However, if v_i and v_j run on different computing nodes, then $c_{e_{i,j}}$ will be affected by many factors, For example, the long $l_{e_{i,j}}$ between two computing nodes, the bandwidth $b_{e_{i,j}}$ of the network, the data rate $r_{e_{i,j}}$ on the directed edge $e_{i,j}$, and can be described as (12):

$$c_{e_{i,j}} = F_{c_e}\left(l_{e_{i,j}}, b_{e_{i,j}}, r_{e_{i,j}}\right). \tag{12}$$

For each data tuple, communication cost $c_{e_{i,j}}$ can be simple calculated by (13):

$$c_{e_{i,j}} = \begin{cases} 0, \textit{if } v_i \textit{ and } v_j \textit{ run on the same computing node}, \\ \frac{r_{e_{i,j}}}{b_{e_{i,j}}}, \textit{otherwise}. \end{cases} \tag{13}$$

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 run on computing node cn_j, respectively. For the source vertex v_s, the earliest start time EST_{v_s} on any computing nodes are shown as (14):

$$EST_{v_s,cn_{any}} = 0. \tag{14}$$

The earliest finish time EFT_{v_s} is the finish time of run source vertex v_s on the computing node with the best processing ability p_{cn_j}, as shown in (15):

$$EFT_{v_s,cn_{p_{best}}} = c_{v_i,cn_{p_{best}}}. \tag{15}$$

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

$$EST_{v_i,cn_j} = \max\left\{t^{idle}_{v_i,cn_j}, t^{data}_{v_{pred},cn_j}\right\}, \tag{16}$$

where $t^{idle}_{v_i,cn_j}$ is the earliest time at which computing node cn_j is ready for use, $t^{data}_{v_{pred},cn_j}$ is the time at which all input data of immediate predecessor vertices of vertex v_i will be ready on computing nodes cn_j, and can be calculated by (17):

$$t^{data}_{v_{pred},cn_j} = \max_{v_{pred}\in pred(v_i)}\left\{EFT_{v_{pred}} + c_{e_{pred,i}}\right\}, \tag{17}$$

where $pred(v_i)$ is the set of immediate predecessor vertices of vertex v_i.

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

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

LST_{v_i,cn_j} is the latest start time of vertex v_i run on computing node cn_j, which can be calculated by traversing the DSG 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 equal to the earliest start time EST_{v_e,cn_j} on computing nodes cn_j, that is.

$$LST_{v_e,cn_j} = EST_{v_e,cn_j}. \tag{19}$$

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 (20):

$$LST_{v_i,cn_j} = \min_{v_{succ}\in succ(v_i)}\left\{LST_{v_{succ}} - c_{e_{i,succ}}\right\} - c_{v_i,cn_j}, \tag{20}$$

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

Definition 8 (*Critical path, CP*). Also called the longest path, CP is a path having the longest latencies from source vertex v_s to end vertex v_e in DSG G, all vertices on CP with the feature of earliest start time EST equal to latest start time LST.

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

Definition 9 (*Critical vertex, CV*). In a big data stream computing environment, CV is a vertex on the critical path, CP, of the DSG G. To the currently situation, if the data stream rate change, the allocating of vertices is not a good choice, moving CV to a new best fit computing node, the maximum reduce response time will be achieved.

The reduced time Δt_{cv} of reschedule CV can be calculated by (21):

$$\Delta t_{cv} = \min\left\{\left|CP_{after} - CP_{before}\right|, \left|c_{v_{cv},cn_{after}} - c_{v_{cv},cn_{before}}\right|\right\}. \tag{21}$$

where CP_{after}, CP_{before} $c_{v_{cv},cn_{after}}$, and $c_{v_{cv},cn_{before}}$ are the CP after reschedule CV, the CP before reschedule CV, the computation cost of CV on before reschedule computing node cn_{before}, and the computation cost of CV on after reschedule 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 be changed. Reschedule all vertices are not a good choice, as those entire data stream will be lost. The history allocated information is not considered, huge system fluctuations is unavoidable. In Re-Stream, during the first placement of DSG, a critical path based earliest completion time priority placement strategy is employed. Hereafter, when arrival rate of data stream is changed, the response time can be improved significantly. A critical vertex on critical path of a DAG is selected and scheduled to a new best fit computing node to achieve minimum response time and system fluctuations.

3.4. Energy consumption model

The energy consumption E_{cn_i} of the ith computing node cn_i at a data center can be calculated by power consumption $P_{cn_i}(\mu_i(t))$ during the time interval $[t_0, t_{n-1}]$ of the ith computing node cn_j at the data center, as shown in (22).

$$E_{cn_i} = \int_{t_0}^{t_{n-1}} P_{cn_i}(\mu_i(t))dt. \tag{22}$$

where power consumption $P_{cn_i}(\mu_i(t))$ during the time interval $[t_0, t_{n-1}]$ of the ith computing node cn_i at the data center can be represented by (23),

$$P_{cn_i}(\mu_i(t)) = P_{cn_i}^{idl}(\mu_i(t)) + P_{cn_i}^{spd}(\mu_i(t)). \tag{23}$$

where $P_{cn_i}^{idl}(\mu_i(t))$ is the power consumption when the ith computing node cn_i at the data center in the idle state, $P_{cn_i}^{spd}(\mu_i(t))$ is the power consumption spreading between idle and full utilized, and $\mu_i(t) \in [0,1]$ is load factor on the ith computing node cn_i at the data center, which changes over time. In today's data centers, load factor $\mu_i(t)$ is usually in the range of [0.1, 0.5].

According to [5], the power consumption $P_{cn_i}(\mu_i(t))$ is mostly determined by the CPU, memory, disk storage and network interfaces. For simplicity, it is calculated by the power consumption of CPU. Meanwhile, the power consumption of application of DVFS [18] on the CPU is almost a linear power-to-frequency relationship for a computing node. So,

$$\begin{cases} P_{cn_i}^{idl}(\mu_i(t)) = \alpha \cdot P_{cn_i}(\max), \alpha \in [0,1], \\ P_{cn_i}^{spd}(\mu_i(t)) = (1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_i(t), \mu_i(t) \in [0,1]. \end{cases} \tag{24}$$

where $P_{cn_i}(\max)$ is the maximum power consumption when the ith computing node cn_i at the data center in the fully utilized state, $\alpha \in [0,1]$ is an adjustable parameter between $P_{cn_i}^{idl}(\mu_i(t))$ and $P_{cn_i}^{spd}(\mu_i(t))$, which is determined by the specific computer architecture.

The energy consumption E_{cn_i} of the ith computing node cn_i at the data center can be further described by Theorem 1.

Theorem 1. If the time interval $[t_0, t_{n-1}]$ can be further divided into $[t_0, t_1, \ldots, t_{n-1}]$, $\forall [t_k, t_{k+1}]$, $k \in [0, 1, \ldots, n-1]$, and $\mu_i(t \in [t_k, t_{k+1}]) \equiv \mu_{i_k} \in [0,1]$, where μ_{i_k} is a constant, and does not change over time, then energy consumption E_{cn_i} can be calculated by (25),

$$E_{cn_i} = (\alpha \cdot P_{cn_i}(\max)) \cdot (t_{n-1} - t_0) + ((1-\alpha) \cdot P_{cn_i}(\max)) \cdot \sum_{k=1}^{n-1}\left(\mu_{i_{k-1}} \cdot (t_k - t_{k-1})\right). \tag{25}$$

Proof. The power consumption $P_{cn_i}(\mu_i(t))$ can be divided into two parts, $P_{cn_i}^{idl}(\mu_i(t))$ and $P_{cn_i}^{spd}(\mu_i(t))$. Therefore,

$$\begin{aligned}
E_{cn_i} &= \int_{t_0}^{t_{n-1}} P_{cn_i}(\mu_i(t))dt = \int_{t_0}^{t_{n-1}} \left(P_{cn_i}^{idl}(\mu_i(t)) + P_{cn_i}^{spd}(\mu_i(t))\right)dt \\
&= \int_{t_0}^{t_{n-1}} P_{cn_i}^{idl}(\mu_i(t))dt + \int_{t_0}^{t_{n-1}} P_{cn_i}^{spd}(\mu_i(t))dt \\
&= \int_{t_0}^{t_{n-1}} \alpha \cdot P_{cn_i}(\max)dt + \int_{t_0}^{t_{n-1}} (1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_i(t)dt \\
&= (\alpha \cdot P_{cn_i}(\max)) \cdot (t_{n-1} - t_0) + \int_{t_0}^{t_{n-1}} (1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_i(t)dt
\end{aligned}$$

When the time interval $[t_0, t_{n-1}]$ is divided into $n-1$ constant time interval $[t_0, t_1, \ldots, t_{n-1}]$, $\forall [t_k, t_{k+1}]$, $i \in [0, 1, \ldots, n-1]$, $\mu_i(t \in [t_k, t_{k+1}]) \equiv \mu_{i_k} \in [0, 1]$, and μ_{i_k} is a constant, then we obtain,

$$\begin{aligned}
E_{cn_i} &= (\alpha \cdot P_{cn_i}(\max)) \cdot (t_{n-1} - t_0) + \int_{t_0}^{t_{n-1}} (1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_i(t)dt \\
&= (\alpha \cdot P_{cn_i}(\max)) \cdot (t_{n-1} - t_0) + \int_{t_0}^{t_1} \left((1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_{i_0}\right)dt + \int_{t_1}^{t_2} \left((1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_{i_1}\right)dt + \ldots \\
&+ \int_{t_{n-2}}^{t_{n-1}} \left((1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_{i_{n-2}}\right)dt \\
&= (\alpha \cdot P_{cn_i}(\max)) \cdot (t_{n-1} - t_0) + \left((1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_{i_0}\right) \cdot (t_1 - t_0) + \left((1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_{i_1}\right) \cdot (t_2 - t_1) + \ldots \\
&+ \left((1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_{i_{n-2}}\right) \cdot (t_{n-1} - t_{n-2}) \\
&= (\alpha \cdot P_{cn_i}(\max)) \cdot (t_{n-1} - t_0) + \sum_{k=1}^{n-1} \left(\left((1-\alpha) \cdot P_{cn_i}(\max) \cdot \mu_{i_{k-1}}\right) \cdot (t_k - t_{k-1})\right) \\
&= (\alpha \cdot P_{cn_i}(\max)) \cdot (t_{n-1} - t_0) + ((1-\alpha) \cdot P_{cn_i}(\max)) \cdot \sum_{k=1}^{n-1} \left(\mu_{i_{k-1}} \cdot (t_k - t_{k-1})\right).
\end{aligned}$$

The proof ends. □

Lemma 1 (*System energy consumption*). System energy consumption E_{sys} is the quantitative denotation of energy consumption of a big data stream computing environment. If a big data stream computing environment, a data centers composed of *num* computer nodes. The system energy consumption E_{sys} during the time interval $[t_0, t_{n-1}]$ can be quantified by (26):

$$E_{sys} = \sum_{i=0}^{num-1} E_{cn_i}. \tag{26}$$

The maximum power consumption $P_{cn_i}(\max)$ of the ith computing node cn_i at the data center in the fully utilized state can be directly measured, according to (25), we have,

$$E_{sys} = \sum_{i=0}^{num-1} \left((\alpha \cdot P_{cn_i}(\max)) \cdot (t_{n-1} - t_0) + ((1-\alpha) \cdot P_{cn_i}(\max)) \cdot \sum_{k=0}^{n-1} \left(\mu_{i_{k-1}} \cdot (t_k - t_{k-1})\right) \right).$$

4. Scheduling algorithm

In this section, we focus our attention on the scheduling algorithm of the Re-Stream in a real big data stream computing platform, Storm. The scheduling algorithm includes first placement of DSG algorithm, performance aware dynamic scheduling of critical vertices algorithm, energy aware dynamic scheduling of non-critical vertices algorithm, and the main scheduling algorithm. All those scheduling algorithms can be implemented by rewrite the IScheduler interface of Storm platform [2,31].

4.1. First placement of DSG

When a DSG is first placed to computing nodes, a critical path based earliest completion time priority placement strategy is employed, in which computing cost and communication cost are considered. In this strategy, the vertices on critical path will be placed, in a high priority, onto the earliest completion computing node only if its predecessors have been placed, if 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.

The first placement of DSG algorithm is described in Algorithm 1.

Algorithm 1. First placement of DSG algorithm.

```
1. Input: DSG $G$, capacity ability vector of computing nodes in data center, and the computation cost matrix $C_{v_{n\times m}}$.
2. Output: Vertices schedule result on computing nodes.
3. if DSG $G$ or computing nodes is null then
4.   Return null.
5. end if
6. Sort all vertices topologically in DSG $G$.
7. Calculate the earliest start time $EST$ and the latest start time $LST$ of each vertex in DSG $G$ by (16) and (20).
8. Determine the critical path $CP$ DSG $G$ according to Definition 8.
9. for each vertex on critical path $CP$ of DSG $G$ do
10.   if vertex $v_i$ with the feature of in degree is zero then
11.     Select vertex $v_i$ as the will be selected vertex $v_{sel}$.
12.   else
13.     Select an immediate predecessor vertex of vertex $v_i$ as the will be selected vertex $v_{sel}$.
14.   end if
15.   Calculate the earliest finish time $EFT_{v_i}$ of vertex $v_i$ on all remaining available computing nodes.
16.   Schedule vertex $v_{sel}$ to the earliest finish computing nodes $cn_j$.
17.   Refresh the capacity ability of computing nodes $cn_j$.
18.   Refresh the in degree of the immediate successor vertices of vertex $v_{sel}$.
19.   Recalculate the earliest start time $EST$ and the latest start time $LST$ of each vertex in DSG $G$ by (16) and (20).
20.   Refresh the critical path $CP$ of the DSG $G$ according to Definition 8.
21. end for
22. return final Vertices schedule strategy on computing nodes.
```

The input of first placement algorithm are DSG G, capacity ability vector of computing nodes in data centers, and the computation cost matrix $C_{v_{n\times m}}$. The output is vertices schedule result on computing nodes. Step 9 to step 21 schedule vertices to computing nodes by critical path based earliest completion time priority scheduling strategy, in which computing cost and communication cost are under consideration.

4.2. Performance aware dynamic scheduling of critical vertices

When arrival rate of data stream is changed, the response time may also be changed. A critical vertex on critical path of a DSG is selected and scheduled to a new computing node to achieve minimum response time and system fluctuations.

The performance aware dynamic scheduling of critical vertices algorithm is described in Algorithm 2.

Algorithm 2. Performance aware dynamic scheduling of critical vertices algorithm.

```
1. Input: DSG $G$, capacity ability vector of computing nodes in data centers, vertex and computing node relationship, and the new input data rate $ir_{v_i}$.
2. Output: Vertices reschedule result on computing nodes.
3. Calculate the earliest start time $EST$ and the latest start time $LST$ of each vertex in DSG $G$ by (16) and (20).
4. Determine the critical path $CP$ DSG $G$ according to Definition 8.
5. Definition maximum reduced time $\Delta t^{\max}_{cv_{\max}} = 0$.
6. for each vertex on critical path $CP$ of DSG $G$ do
7.   Fetch a vertex $v_i$.
8.   for each immediate predecessor or successor vertex of vertex $v_i$ do
9.   Try to reschedule vertex $v_i$ to computing nodes of the immediate predecessor or successor vertices with the constraint of remaining available capacity ability is enough.
10.   Calculate this new reduced time $\Delta t^{new}_{cv_i}$.
11.     if the new reduced time $\Delta t^{new}_{cv_i}$ is bigger than $\Delta t^{\max}_{cv_{\max}}$ then
12.       $\Delta t^{\max}_{cv_{\max}} = \Delta t^{new}_{cv_i}$.
13.       $cv_{\max} = v_i$.
14.       $cn_{\max} = cn^{try}_{v_i}$.
15.     end if
16.   end for
17. end for
18. Recalculate critical vertex $cv_{\max}$ to the computing node $cn_{\max}$.
19. return final Vertices schedule strategy on computing nodes.
```

The input of performance aware dynamic scheduling of critical vertices algorithm are DSG G, capacity ability vector of computing nodes in data centers, vertex and computing node relationship, and the new input data rate ir_{v_i}. The output is vertices reschedule result on computing nodes. Step 6 to step 17 determine a critical vertex $cv_{\max}$ on critical path of a DAG and scheduled to a target computing node $cn_{\max}$.

4.3. Energy aware dynamic scheduling of non-critical vertices

An energy model is employed to estimate the energy consumption of each computing node in the current data stream environments. An energy-efficient consolidation of non-critical vertices on non-critical path strategy is employed to maximize the energy efficiency without disturbing the response time of the current DSG. Usually, the more the vertices of a DSG, the higher the energy efficiency will be achieved. The lower the CUP utilization of each computing node, the higher the energy efficiency will be achieved.

The energy aware dynamic scheduling of non-critical vertices algorithm is described in Algorithm 3.

Algorithm 3. Energy aware dynamic scheduling of non-critical vertices algorithm.

```
1. Input: DSG $G$, capacity ability vector of computing nodes in data centers, and vertices schedule result on computing nodes.
2. Output: Vertices reschedule result on computing nodes and system energy consumption $E_{sys}$.
3. Sort all vertices topologically in DSG $G$.
4. Calculate the $EST$ and the $LST$ of each vertex in DSG $G$ by (16) and (20).
5. Determine the critical path $CP$ DSG $G$ according to Definition 8.
6. Sort computing nodes by the remaining available capacity ability in increasing order.
7. for each computing node in data centers do
8.   if no vertex running on computing node $cn$ is vertex on critical path $CP$ then
9.     Identify computing node $cn$ as the energy-aware computing node.
10.  end If
11. end for
12. for each energy-aware computing node in data centers do
13.   Identify energy-aware computing node $cn$ as reschedule computing node.
14.   for each vertex running on energy-aware computing node $cn$ do
15.     Try to reschedule vertex $v_i$ to computing nodes of running the immediate predecessor or successor vertices with the constraint of remaining available capacity ability is enough.
16.     Calculate the length of critical path, $CP$, after reschedule vertex $v_i$
17.     if the length of critical path, $CP$, after reschedule vertex $v_i$ is bigger than before reschedule then
18.       Identify energy-aware computing node $cn$ as un-reschedule computing node.
19.     end if
20.   end for
21.   if energy-aware computing node $cn$ is reschedule computing node then
22.     Schedule all vertices running on energy-aware computing node $cn$ to the corresponds computing nodes of running the immediate predecessor or successor vertices with the constraint of remaining available capacity ability is enough
23.   end if
24. end for
25. Calculate the energy consumption of each computing node by (25).
26. Calculate system energy consumption $E_{sys}$ by (26).
27. return final Vertices schedule strategy on computing nodes and system energy consumption $E_{sys}$.
```

The input of energy aware dynamic scheduling of non-critical vertices algorithm are DSG G, capacity ability vector of computing nodes in data centers, and vertices schedule result on computing nodes. The output are vertices reschedule result on computing nodes and system energy consumption E_{sys}. Step 7 to step 11 identify the energy-aware computing nodes. Step 12 to step 24 reschedule all vertices running on non-critical vertex based computing nodes, and maximize the energy efficiency while not increasing the response time of current DSG.

4.4. Main scheduling algorithm

To achieve high energy efficiency and low response time objectives in big data stream computing environments, we employ Re-Stream in Storm platform. To maximize energy efficiency without affecting the minimization of response time, all vertices in DSG are divided into two groups, vertices on critical path, and vertices on non-critical path. Minimal response time is determined by all the vertices on critical path. Moreover, maximizing energy efficiency can be achieved by consolidation recourse for vertices on non-critical path.

The main scheduling algorithm of Re-Stream is described in Algorithm 4.

Algorithm 4. Main scheduling algorithm.

```
1. Input: DSG G, capacity ability vector of computing nodes in data centers, and data rate of data stream in real-time.
2. Output: Real-time energy efficiency and response time objectives.
3. Evaluate the most commonly data rate in a big data stream computing environments.
4. Calculate the computation cost of each vertex running on each computing node, relevant communication cost of the each edge between corresponding vertices, and get the computation cost matrix $C_{v_{n\times m}}$.
5. First place vertices of DSG to compute nodes in data center by Algorithm 1.
6. Monitor the real-time rate of data stream in the input interface.
7. while $|r_{new} - r_{old}| > \Delta r$ do
8.    Recalculate the computation cost of each vertex running on each node under the new data rate, and relevant communication cost of the each edge between corresponding two vertices, and get the computation cost matrix $C_{v_{n\times m}}$.
9.    Real-time dynamic scheduling critical vertex on critical path ($CP$) to achieve a new low response time objective by Algorithm 2.
10.   Real-time dynamic scheduling vertex on non-critical path to achieve a new high energy efficiency objective by Algorithm 3.
11.   Recalculate the response time of DSG by (18) and the system energy consumption $E_{sys}$ of data center by (26).
12. end while
13. return real-time energy efficiency and response time objectives.
```

The input of main scheduling algorithm are DSG *G*, capacity ability vector of computing nodes in data centers, and data rate of data stream in real-time. The outputs are real-time energy efficiency and response time objectives. Step 7 to step 12 real-time adjust vertices scheduling strategy according to the change of rate of data stream, and to maximize the energy efficiency and minimize response time, to achieve high energy efficiency and low response time objectives trade off in big data stream computing environments.

5. Performance evaluation

To evaluate the performance of the proposed Re-Stream, experimental environment and parameter set are firstly discussed in this section, followed by the precise experimental evaluation results.

5.1. Experimental environment and parameter setup

Storm platform [2,6,28,31] is used in the simulation 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.

16 virtual machines are created in a data center in the simulation environment. Each virtual machine is a dual 4-core Intel Xeon 3 Ghz 32 bit 32 GB Memory, 1 Gbps network interface. Each machine runs Linux Ubuntu Server 13.04. The following software components were used: Java 1.7 OpenJDK Runtime Environment, ZeroMQ 2.1.7, Zookeeper 3.4.5, JZMQ, Python 2.7.2, unzip and Kestrel 2.3.4. Storm 0.8.1 is used.

Moreover, a critical path sensitive DSG is submitted to the data centers, as shown in Fig. 9, the length of 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.

5.2. Experimental results

The experimental set up contained two evaluation parameters: response time and system energy consumption E_{sys}.

(1) *Response Time (RT)*. The response time of a DSG is determined by the critical path of that DSG. RT can be calculated by *EST* of the last vertex on critical path, as show in (18). RT can also be counted by Storm UI, which is providing by Storm platform.
As shown in Fig. 10, when the rate of data stream is unchanged over time, the response time of DSG in Fig. 10 is also fixed at certain stable level. For example, when the input rate of data stream is 10 tuple/s, 20 tuple/s, 50 tuple/s, the

response time is 96 ms, 97 ms, and 126 ms, respectively.

As shown in Fig. 11, when the rate of data stream is changed over time, e.g., the input rate of data stream is 10 tuple/s in interval [0,300] second, 20 tuple/s in interval [300,400] second, 50 tuple/s in interval [400,500] second, the corresponding response time is also changed over data stream. Compared with the default scheduling, round-robin strategy, in Storm platform, our Re-Stream has a better RT.

(2) *System energy consumption*. System energy consumption E_{sys} reflects the total energy consumption of the data center to process the same data stream by a DSG. E_{sys} can be calculated by summing the energy consumption of each computing node, that run the vertices of the DSG. The energy consumption of each computing node can be calculated by (25).

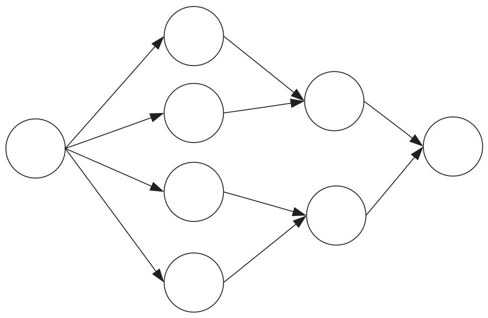
Fig. 9. Critical path sensitive DSG.

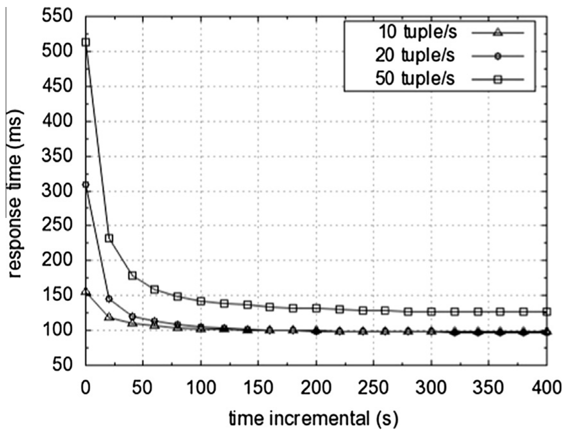

Fig. 10. Response time of DSG in Fig. 9 with stable data stream rate.

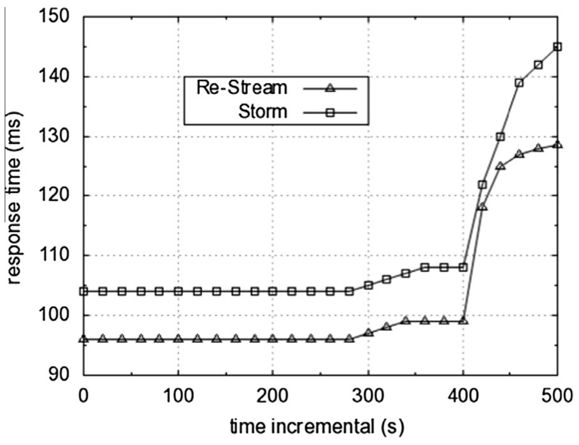

Fig. 11. Response time of DSG in Fig. 9 with changeable data stream rate.

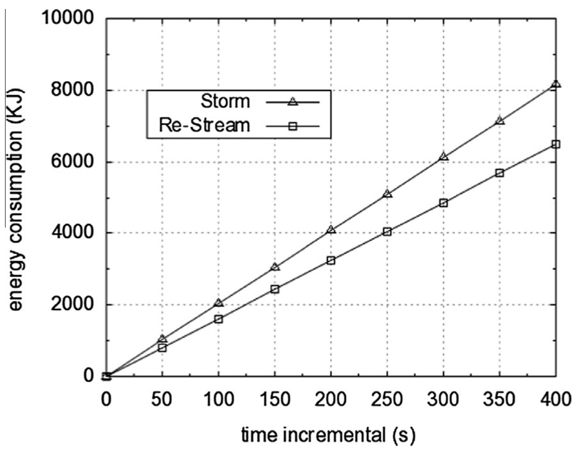

Fig. 12. System energy consumption of DSG in Fig. 9 with stable data stream rate 50 tuple/s.

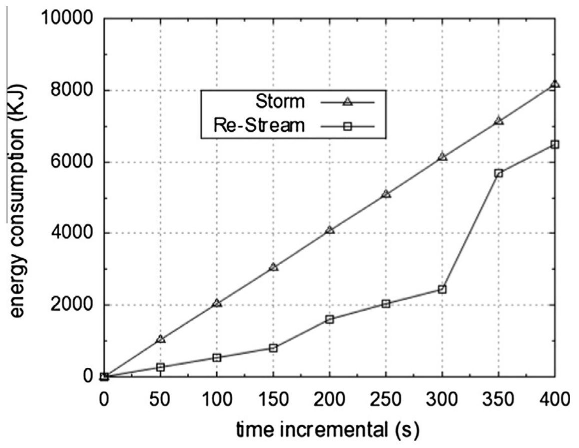

Fig. 13. System energy consumption of DSG in Fig. 9 with changeable data stream rate from 10 to 50 tuple/s.

As shown in Fig. 12, when the rate of data stream is stable at 50 tuple/s, the computing nodes employed by Re-Stream to process data stream is reduced compared with that employed by the default scheduling, round-robin strategy, in the Storm platform. The corresponding energy consumption of DSG in Fig. 9 by Re-Stream is also smaller than that of Storm platform, the longer the more obvious.

As shown in Fig. 13, when the rate of data stream is changed over time, e.g., the input rate of data stream is 10 tuple/s in interval [0,150] second, 20 tuple/s in interval [150,300] second, 50 tuple/s in interval [300,400] second, the corresponding computing node and energy consumption are also changed over data stream. Compared with the default scheduling, round-robin strategy, in the Storm platform, our Re-Stream requires less computing nodes to process the data stream of DSG in Fig. 9 with less energy consumption.

6. Related work

In this section, three broad categories of related work are presented: big data stream computing, performance aware data stream scheduling, and energy aware data stream scheduling.

6.1. 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 model of straight through computing, such as S4 [20], Storm [31], while big data bath computing is model of store then computing, such as MapReduce framework [40], open sourced by the Hadoop implementation [24]. However, those big data batch computing are not adequate for supporting big data stream computing.

In [19], a distributed system, Naiad, was designed specifically for executing cyclic dataflow programs in parallel. 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 [22], a distributed-system infrastructure, *TimeStream*, has been designed specifically for reliable low-latency continuous computing of big streaming data on large clusters. The big streaming data has the following characteristics: (1) high volumes – the incoming data arrives continuously at volumes that far exceed the capabilities of individual machines; (2) low latency – input streams incur multi-staged computing at low latency to produce output streams. A powerful abstraction, resilient substitution, serves as a uniform foundation for handling failure recovery and dynamic reconfiguration correctly and efficiently.

In [39], 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 larger scales data stream computing environments. D-Streams enable a parallel recovery mechanism that improves efficiency over traditional replication and backup schemes, and tolerates stragglers, the ideal in D-Streams is to structure a streaming computation as a series of stateless, deterministic batch computations on small time intervals. D-Streams make (1) the state at each time step 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 is done in big data batch computing, while the response time of batch computing at the level of several seconds or even minutes, which cannot meet the request of most of big data. However, the research about big data stream computing is not enough and systematic. For example, Storm system employs round-robin strategy as its task scheduling strategy, which is not efficient and effective always. Therefore, it needs to pay more attention to stream computing in big data and cloud era.

6.2. Performance aware data stream scheduling

In distributed stream computing environments, the structure of data stream is described as DAG. Performance oriented data stream scheduling is trying to dynamically schedule recourse for many DAGs according the dynamic changes in the volume of data stream, which is NP-hard. These performance oriented data stream scheduling can be further classified into the following groups [32]: list scheduling algorithm, task duplication-based scheduling algorithm, clustering scheduling algorithm, and guided random search scheduling algorithm.

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 [1], a density-based data streams clustering algorithms is surveyed. In [36], a data stream clustering algorithm, 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 [7], a two-phase list-based scheduling algorithm, known as 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 technique to reduce the necessary communication between the processors. Some crucial tasks will be duplicated and executed on more than one processors. In [29], 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.

In [38], 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 are not limited in one or other aspect. Up to now, most of the researches are done in static scheduling. All the information about scheduling is estimated, unchanged, and must be in advance. However, when the volume of data stream is changed, the scheduling may not be a wise strategy. If static scheduling is employed in the new station, huge fluctuations will occur. In big data stream environments, the volume of data stream is changed.

6.3. Energy aware data stream scheduling

In distributed stream computing datacenter, energy savings is becoming a common trend. Energy consumption has become a key metric for evaluating how good a computing system is. Evaluations and comparisons of different energy-aware scheduling techniques were surveyed in [11,34,41]. Generally, energy aware data stream scheduling can be classified into the following three groups: hardware-based energy aware data stream scheduling (Dynamic Power Management and Dynamic Voltage/Frequency Scaling), software-based energy aware data stream scheduling (virtual machine consolidation), and application-based energy aware data stream scheduling (task duplication).

In [4], an energy-aware DAG scheduling, EADAGS, on heterogeneous processors is proposed. EADAGS combines Dynamic Voltage Scaling (DVS) with Decisive Path Scheduling (DPS) to achieve the twin objectives of minimizing finish time and energy consumption. In the first phase, after DPS is run on the DAG to provide a low finish time, the energy consumed is estimated for all processors. In the second phase, voltage scaling is applied during slack times to reduce energy while maintaining the schedule length.

In [12], an Energy-aware Task Consolidation (ETC) technique is proposed. ETC achieves energy-aware task consolidation by restricting CPU use below a specified peak threshold. ETC does energy-aware task consolidation by consolidating tasks amongst virtual clusters. In addition, the energy cost model considers network latency when a task migrates to another virtual cluster.

In [16], a model for estimating the energy consumption of each virtual machine, a virtual machine scheduling algorithm that provides computing resources according to the energy budget of each virtual machine, is proposed. Those schemes are implemented in the Xen virtualization system.

In [43], two energy-efficient duplication-based scheduling algorithms, Energy-Aware Duplication (EAD) and Performance-Energy Balanced Duplication (PEBD), are proposed. The idea of conserving energy is to immediately turn processors to the lowest voltage once no task is waiting or no task is ready for execution. This policy ensures that tasks can be executed as fast as possible. Meanwhile, tasks on the critical path will be duplicated under the condition that no significant energy overhead is introduced by the replicas. Duplications can avoid the performance degradation caused by waiting messages. The rationale behind this approach is twofold. First, energy overhead incurred by task replicas could be offset by energy savings in interconnects and by shortening schedule length. Second, the overall performance is improved by the virtue of replicas.

Existing energy efficient measurement and management mechanism proposed for distributed stream computing system cannot be directly implemented for big data stream computing. As they just base on specific assumptions, focus on minimizing energy consumption, or try to balance energy and performance. In contrast, our solution can save energy consumption without affecting minimization of response time. All vertices in DAG are divided into two groups, vertices on critical path, and vertices on non-critical path. Low response time is only determined by all the vertices on critical path. Moreover, minimizing energy consumption costs can be achieved by consolidation recourse for vertices on non-critical path. Our solution intends to achieve high throughput objectives measurement and management in big data stream computing environments.

7. Conclusions and future work

High energy efficiency issue is one of the major obstacles for opening up the new era of green stream computing. To achieve green stream computing in big data environments, it is important to obtain a clear picture of the total energy consumption in big data environments. Importantly, it is needed to understand how to minimize energy consumption in data centers, and to deal with the high energy efficiency and low response time objectives is missing in most existing research. This justifies the importance of modeling an energy efficient resource scheduling and optimization framework for big data stream computing environments, so as to maximize energy efficiency and minimize response time, and to achieve high energy efficiency and low response time objectives measurement and management in big data stream computing environments.

In this paper, we covered all four aspects of Re-Stream by answering the following questions:

(1) Profiling the mathematical relationship between energy consumption, response time, and resource utilization in big data stream computing, and obtaining the conditions to meet the high energy efficiency and low response time objectives.
(2) Modeling DSG in big data stream computing environments by referring to the distributed stream computing theories, identifying the critical path in DSG, and getting energy consumption of an allocation scheme for a DSG at a special data stream speed.
(3) Allocating tasks by the energy efficient heuristic critical path scheduling mechanism subject to the constraints of tasks' architectural requirements, and online optimizing the scheduling mechanism by reallocating the critical vertices on critical path of DSG to minimize response time and system fluctuations, and consolidating non-critical vertices on non-critical path to maximize the energy efficiency, to meet high energy efficiency and low response time.
(4) Evaluating the high energy efficiency and low response time objectives in big data stream computing environments, and considering both high energy efficiency and low response time.

Our contributions are summarized as follows:

(1) Formal definitions of task topology, big data stream computing architecture, and high energy efficiency and low response time in quantitative perspective, and systematic profiling of the mathematical relationship between energy consumption, response time, and resource utilization in big data stream.

(2) Systematic investigation in computation of nodes, communication among nodes, and allocation of tasks by the energy efficient heuristic critical path scheduling mechanism subject to the constraints of tasks' architectural requirements.
(3) Identification of the critical path of a DSG at a special data stream throughput, and online optimization of scheduling mechanism by reallocating the critical vertices on critical path of DSG to minimize response time and system fluctuations.
(4) Quantification of energy consumption of an allocation scheme for a DSG at a special data stream speed, and consolidation of non-critical vertices on non-critical path to maximize the energy efficiency, to meet high energy efficiency objective.
(5) Prototype implementation, simulation, and performance evaluation of the proposed Re-Stream, that can construct high energy efficiency and low response time trade-off effectively in big data stream computing environments.

Future works will focus on the following:

(1) Optimizing the structure of DSG, and providing an efficient DSG for each application, and providing a dynamical instance of each vertex.
(2) Examining the efficiency and effectiveness of Re-Stream in real big data stream computing environments.
(3) Developing a complete green stream computing framework based on Re-Stream as a part of big data stream computing services to satisfy the high energy efficiency and low response time objectives.
(4) Deploying the Re-Stream on real big data stream computing environments.

Acknowledgements

This work was supported by the National Natural Science Foundation of China Grants 61170008 and 61272055, the National Fundamental Research Program of China 973 Program, 2014CB340402, and the China Postdoctoral Science Foundation Grant 2014M560976. This research was also in part supported by a Grant from the National Science Foundation, CNS 1229316.

References

[1] A. Amini, T.Y. Wah, H. Saboohi, On density-based data streams clustering algorithms: a survey, J. Comput. Sci. Technol. 29 (1) (2014) 116–141.
[2] L. Aniello, R. Baldoni, L. Querzoni, Adaptive online scheduling in storm, in: Proc. 7th ACM International Conference on Distributed Event-Based Systems, DEBS 2013, ACM Press, 2013, pp. 207–218.
[3] J. Baliga, R.W.A. Ayre, K. Hinton, R.S. Tucker, Green cloud computing: balancing energy in processing, storage, and transport, Proc. IEEE 99 (1) (2011) 149–167.
[4] S. Baskiyar, R. Abdel-Kader, Energy aware DAG scheduling on heterogeneous systems, Cluster Comput. 13 (4) (2010) 373–383.
[5] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.
[6] J. Chauhan, S.A. Chowdhury, D. Makaroff, Performance evaluation of Yahoo! S4: A first look, in: Proc. 7th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2012, IEEE Press, 2012, pp. 58–65.
[7] M.I. Daoud, N. Kharma, A hybrid heuristic-genetic algorithm for task scheduling in heterogeneous processor networks, J. Parall. Distr. Comput. 71 (11) (2011) 1518–1531.
[8] M. Dayarathna, S. Toyotaro, Automatic optimization of stream programs via source program operator graph transformations, Distr. Parall. Databases 31 (4) (2013) 543–599.
[9] H. Demirkan, D. Delen, Leveraging the capabilities of service-oriented decision support systems: putting analytics and big data in cloud, Decis. Support Syst. 55 (1) (2013) 412–421.
[10] G. Guo, Y. Zhao, G.Q. Yang, Cooperation of multiple mobile sensors with minimum energy cost for mobility and communication, Inform. Sci. 254 (2014) 69–82.
[11] G.J. Hornga, T.Y. Changb, S.T. Chengc, An effective node-selection scheme for the energy efficiency of solar-powered WSNs in a stream environment, Exp. Syst. Appl. 41 (7) (2014) 3143–3156.
[12] C.H. Hsu, K.D. Slagter, S.C. Chen, Y.C. Chung, Optimizing energy consumption with task consolidation in clouds, Inform. Sci. 258 (2014) 452–462.
[13] J.W. Jang, M. Jeon, H.S. Kim, H. Jo, J.S. Kim, S. Maeng, Energy reduction in consolidated servers through memory-aware virtual machine scheduling, IEEE Trans. Comput. 60 (4) (2011) 552–564.
[14] S.U. Khan, I. Ahmad, A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids, IEEE Trans. Parall. Distr. Syst. 20 (3) (2009) 346–360.
[15] R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, J. Wolf, K.L. Wu, H. Andrade, B. Gedik, COLA: optimizing stream processing applications via graph partitioning, in: Proc. 10th ACM/IFIP/USENIX International Conference on Middleware, Middleware 2009, ACM Press, 2009, pp. 1–20.
[16] N. Kima, J. Chob, E. Seob, Energy-credit scheduler: an energy-aware virtual machine scheduler for cloud systems, Future Gener. Comput. Syst. 32 (3) (2014) 128–137.
[17] J. Lu, D. Li, Bias correction in a small sample from big data, IEEE Trans. Knowl. Data Eng. 25 (11) (2013) 2658–2663.
[18] D.S. Ma, R. Bondade, Enabling power-efficient DVFS operations on silicon, IEEE Circ. Syst. Mag. 10 (1) (2011) 14–30.
[19] 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, 2013, pp. 439–455.
[20] 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, 2010, pp. 170–177.
[21] M. Pedram, Energy-efficient datacenters, IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 31 (10) (2012) 1465–1484.
[22] 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, 2013, pp. 1–14.
[23] M. Rahman, R. Hassan, R. Ranjan, R. Buyya, Adaptive workflow scheduling for dynamic grid and cloud computing environment, Concur. Comput.-Pract. Exp. 25 (13) (2013) 1816–1842.

[24] 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, 2013, pp. 402–411.
[25] H. Shao, L. Rao, Z. Wang, X. Liu, Z. Wang, Kui Ren, Optimal load balancing and energy cost management for Internet data centers in deregulated electricity markets, IEEE Trans. Parall. Distr. Syst. 25 (10) (2014) 2659–2669.
[26] M. Sharifi, S. Shahrivari, H. Salimi, PASTA: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources, Computing 95 (1) (2013) 67–88.
[27] S.J. Shin, D.S. Lee, W.S. Lee, CP-tree: an adaptive synopsis structure for compressing frequent itemsets over online data streams, Inform. Sci. 278 (2014) 559–576.
[28] D. Simoncelli, M. Dusi, F. Gringoli, S. Niccolini, Stream-monitoring with BlockMon: convergence of network measurements and data analytics platforms, ACM Sigcomm Comput. Commun. Rev. 43 (2) (2013) 29–35.
[29] O. Sinnen, A. To, M. Kaur, Contention-aware scheduling with task duplication, J. Parall. Distr. Comput. 71 (1) (2011) 77–86.
[30] G.L. Stavrinides, H.D. Karatza, Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques, Simul. Model. Practice Theory 19 (1) (2011) 540–552.
[31] Storm. <http://storm-project.net/>.
[32] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, J. Wang, Cost-efficient task scheduling for executing large programs in the cloud, Parall. Comput. 39 (4–5) (2013) 177–188.
[33] J.M. Tien, Big data: unleashing information, J. Syst. Sci. Syst. Eng. 22 (2) (2013) 127–151.
[34] R. Tu, X. Wang, Y. Yang, Energy-saving model for SDN data centers, J. Supercomput. 70 (3) (2014) 1477–1495.
[35] H. Vinay, V. Sarma, Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling, IEEE Trans. Comput. 63 (2) (2014) 349–360.
[36] 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) (2013) 1410–1424.
[37] 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, 2014, pp. 535–544.
[38] 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. Parall. Distr. Comput. 73 (9) (2013) 1306–1322.
[39] 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, 2013, pp. 423–438.
[40] 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, 2013, pp. 35–39.
[41] S. Zhuravlev, J.C. Saez, S. Blagodurov, A. Fedorova, M. Prieto, Survey of energy-cognizant scheduling techniques, IEEE Trans. Parall. Distr. Syst. 24 (7) (2013) 1447–1464.
[42] M. Zihayat, A. An, Mining top-k high utility patterns over data streams, Inform. Sci. 285 (2014) 138–161.
[43] Z.L. Zong, A. Manzanares, X.J. Ruan, X. Qin, EAD and PEBD: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters, IEEE Trans. Comput. 60 (3) (2011) 360–374.