IEICE TRANS. INF. & SYST., VOL.E95–D, NO.4 APRIL 2012

PAPER

Economical and Fault-Tolerant Load Balancing in Distributed Stream Processing Systems

Fuyuan XIAO[†], *Nonmember*, **Teruaki KITASUKA**[†], *Member*, *and* **Masayoshi ARITSUGI**[†a)], *Senior Member*

SUMMARY We present an economical and fault-tolerant load balancing strategy (EFTLBS) based on an operator replication mechanism and a load shedding method, that fully utilizes the network resources to realize continuous and highly-available data stream processing without dynamic operator migration over wide area networks. In this paper, we first design an economical operator distribution (EOD) plan based on a bin-packing model under the constraints of each stream bandwidth as well as each server's CPU capacity. Next, we devise super-operator (SO) that load balances multi-degree operator replicas. Moreover, for improving the fault-tolerance of the system, we color the SOs based on a coloring bin-packing (CBP) model that assigns peer operator replicas to different servers. To minimize the effects of input rate bursts upon the system, we take advantage of a load shedding method while keeping the QoS guarantees made by the system based on the SO scheme and the CBP model. Finally, we substantiate the utility of our work through experiments on ns-3.

key words: *distributed data stream processing systems (DDSPSs), load balancing strategy, fault-tolerance, economical operator distribution (EOD)*

1. Introduction

Nowadays, there has been increasing interest in handling high volume data and continuous data streams with low latency on distributed stream processing systems (DSPSs) [1]–[6]. In such systems, data streams are processed in or near real-time for a variety of purposes, such as network monitoring, intrusion detection, real-time analysis and customized e-commerce applications. In those application domains, continuous and highly-available data stream processing with low latency is critical for dealing with real-world events.

In DSPSs, however, bursts in data may cause bottlenecks at some points along the server chain. Bottlenecks slow down the data stream processing and network transmission, and result in high latency. There are several kinds of load balancing strategies to deal with the above problems. According to the classification of load management strategy's attribution, there are three kinds of techniques. One is dynamic load management [7]–[10] in which operators can be moved in the case of servers overload during the data stream processing. Needless to say, operator movement is too expensive to alleviate short-term bursts; moreover, some systems do not support the ability to move operators dynamically. As a result, dealing with short-term load fluctuations by frequent operator re-distribution is typically prohibited. Another is static load management, such as resilient operator distribution (ROD) [11]. ROD, a static load management method, can provide an elastic initial distribution plan to avoid overload for input rates without operator movement. However, such an approach does not regard energy consumption. Although there are many resources over the wide area network, we should take energy consumption into consideration that is a crucial and rising problem in the real world [12], namely, fully utilizing the network resources for decreasing resources waste. The other is the load shedding method [13]–[15] that drops part of data once servers are overloaded during the data stream processing. In those load shedding techniques, however, they just take the load balancing of a single-degree operator replication into account while selectively dropping some data for load balancing. Such a behavior gives rise to the poor quality of query results and loses the QoS guarantees of the system. Moreover, the previous system models [8]–[11], [13]–[15], are based on the assumption that the bandwidth is not a bottleneck. In the real world, however, it may cause overload on the network during the stream processing.

To overcome the limitations of the previous techniques, we propose a novel approach, an economical and fault-tolerant load balancing strategy (EFTLBS). The EFTLBS aims to fully utilize the network resources to realize continuous and highly-available data stream processing without dynamic operator migration over wide area networks. In our method, for initializing a good economical operator distribution (EOD) plan before the data stream processing, we design an operator distribution plan based on a bin-packing (BP) model that is expressed as a set of linear functions of input stream rates (illustrated in Sect. 2.3). In addition, Xing et al. [11] proposed that dealing with the "high impact" operators late may cause the system to significantly deviate from the optimal results. Conclusively, the load balancing of multi-degree operator replicas for initializing an operator distribution plan takes an essential effect in the performance of the system. Hence, we devise super-operator (SO) that load balances multi-degree operator replicas for initializing an operator distribution plan.

Then for improving the fault-tolerance of the system, we also develop coloring super-operator (CSO) based on a coloring bin-packing (CBP) model. The CSO assigns peer operator replicas to different servers to overcome the problems of server failures (illustrated in Sect. 3). To minimize

Manuscript received July 15, 2011.
Manuscript revised November 22, 2011.

[†]The authors are with the Graduate School of Science and Technology, Kumamoto University, Kumamoto-shi, 860–8555 Japan.
a) E-mail: aritsugi@cs.kumamoto-u.ac.jp

DOI: 10.1587/transinf.E95.D.1062

the effects of input rate bursts upon the system without operator movement during the data stream processing, we take full advantage of a load shedding method while keeping the QoS guarantees made by the system based on a multi-degree operator replication mechanism and the CBP model (illustrated in Sect. 4). Through seamlessly combining the above methods, namely, the EOD approach, the CSO scheme and the load shedding method, we can realize continuous and highly-available data stream processing.

In this paper, we also model the bandwidth capacity of each stream as well as each server's CPU capacity as the constraints in our economical operator distribution (EOD) model. More specifically, we express the load of each stream and the load of each operator as a function of input stream rates of the system based on the bin-packing model. For given input stream rates and a given operator assignment plan, the system is feasible, i.e., none of the nodes and bandwidth of the streams are overloaded. Our goal is to find a minimum energy consumption scheme in which a system stars using the minimization of the number of servers, in other words, having maximum server utilization for decreasing the network resources waste. We call such a technique as an economical and fault-tolerant load balancing strategy (EFTLBS).

1.1 Challenges

In this paper, we consider stream processing in a wide area network that spans diverse areas of the globe. To realize correct and timely processing, we must address the following challenges [11], [16]:

- In a wide area network, as we use more servers, server failures are more likely to occur. A failed server cannot send data that gives rise to the poor quality of query results and loses the QoS guarantees of the system.
- Computer networks are vulnerable to link failures and congestion. The communication outages sometimes cause high latency.
- A server can be overloaded due to the input rate bursts upon the system or by other applications that share the server. In this case, the stream processing at subsequent servers also gets delayed.

1.2 Contributions

To overcome the limitations of the previous techniques based on the above challenges, we propose the EFTLBS. The contributions of this paper are as follows:

- We propose a new concept of load balancing, economical and fault-tolerant load balancing, to realize continuous and highly-available stream processing over a wide area network.
- We propose a global load management strategy to ensure the normal operation of the system not only for operator distribution at the beginning of the system running but also for controlling overloaded nodes while the system runs.
- We develop coloring super-operator (CSO) based on a coloring bin-packing (CBP) model that assigns peer operator replicas to different servers to improve the fault-tolerance of the system.
- We take full advantage of a load shedding method based on a multi-degree operator replication mechanism and the CBP model to realize static load balancing and keep the QoS guarantees of the whole system without dynamic operator migration among nodes.
- We propose modeling load balancing as a bin packing optimization problem under the limitations of each stream bandwidth as well as each computer's CPU capacity.

1.3 Organization

The rest of this paper is organized as follows. In Sect. 2, we first show the assumptions of our load model. Next, we design an economical operator distribution (EOD) plan and formalize the EOD problem for the system based on a bin-packing pattern. We also model the bandwidth capacity of each stream as well as each server's CPU capacity as the constraints in our EOD plan. In Sect. 3, we devise super-operator (SO) and color each SO based on a coloring bin-packing (CBP) model for overcoming the problems of server failures. In addition, in Sect. 4, we also model the load shedding problem based on the SO scheme and the CBP model to overcome the problems of input rate bursts. In Sect. 5, we seamlessly combine the EOD approach, the CSO scheme, and the load shedding method as an economical and fault-tolerant load balancing strategy (EFTLBS) to achieve continuous and highly-available data stream processing with low latency over wide area networks. Then, we show the experimental results in Sect. 6. We present the related work in Sect. 7 and conclude in Sect. 8.

2. Economical Operator Distribution Plan

In this section, we first describe the assumptions behind our work and introduce some key definitions and notations. Next, we describe our economical operator distribution model under the limitations of each server's CPU capacity and each stream bandwidth capacity in detail. Then, we formally state the corresponding EOD model as a bin-packing (BP) optimization problem.

2.1 Assumptions

- **System Configuration.** We assume a wide area network that consists of loosely coupled, shared-nothing computers as the substructure for the stream processing in which multi-degree operator replicas run in parallel and independently [16].
- **Communication.** We assume that the servers are con-

nected with a fast, reliable, order-preserving and robust message delivery network protocol such as TCP. We further assume that the bandwidth is a limited resource but the network transfer delays as well as the CPU overhead for the data stream transfer are negligible.

- **Query.** The operators to be distributed on the servers are connected in a data-flow-style acyclic query graph. The graph is generally referred to as a directed acyclic graph (DAG). In this paper, we take each continuous query operator as the minimum task allocation unit.
- **Failure Model.** We assume there are fail-stop server/network failures. We do not consider Byzantine failures [17].

2.2 Definitions and Notations

We now introduce some key notations and definitions that are used in the remainder of this paper (shown in Table 1).

We represent the distribution of operators on nodes of the system by the following operator allocation matrix:

$$A = \{a_{ij}\}_{n\times m}$$

where $a_{ij} = 1$ if operator o_j is assigned to node N_i and $a_{ij} = 0$ otherwise while $\Sigma_{i=1}^{n} a_{ij}$ represents the replication degree.

We represent the connection among operators by the following operator connection matrix:

$$B^o = \{b^o_{jh}\}_{m\times m}$$

where $b^o_{jh} = 1$ if operator o_j is connected to operator o_h and $b^o_{jh} = 0$ otherwise.

We also represent the connection among nodes that is used for stream processing by the following node connection used for the stream processing matrix:

$$B^n = \{b^n_{is}\}_{n\times n}$$

where $b^n_{is} = 1$ if and only if there exists at least one stream between nodes N_i and N_s, i.e., $\Sigma_{j=1}^{m-1}\Sigma_{h=j+1}^{m} a_{ij}b^o_{jh}a_{sh} > 0$ and $b^n_{is} = 0$ otherwise.

2.3 Load Model

We define the EOD problem based on a bin-packing (BP) model as follows. Consider a query diagram as shown in Fig. 1. We assume there are n nodes ($N_i, i = 1, \cdots, n$), each with a fixed dedicated CPU capacity $N^c_i (i = 1, \cdots, n)$, m operators ($o_j, j = 1, \cdots, m$), d input streams ($I_k, k = 1, \cdots, d$), the input stream rate of each of which is represented as r_k in the system, and each bandwidth between nodes N_i and N_s with a fixed dedicated bandwidth capacity B^c_{is}. Generally, an operator may have multiple input streams and multiple output streams. The rate of a stream is defined as the number of data items (tuples) that arrive at the stream per second. We define c_j as the cost of an operator o_j in terms of the average processing times (seconds) for processing an input tuple with respect to input streams. The selectivity of an operator, s_j, is defined as the ratio of the output stream rates to the input stream rates: $\#output/\#input$ ($s_j, j = 1, \cdots, m$). We denote the selectivity for the k^{th} input stream to the down operator o_j by s_{kj} ($k = 1, \cdots, d, j = 1, \cdots, m$). We define the load of each operator in terms of a function of operator costs, selectivities and the system input stream rates. We define the bandwidth of each stream as the number of data items (tuples) that are transferred per second, that is, as a

Table 1 Notation.

Notation	Meaning
n	number of nodes
m	number of operators
d	number of the system input streams
N_i	the i^{th} node
o_j	the j^{th} operator
I_k	the k^{th} input stream
r_k	the k^{th} input stream rate
U^c_i	CPU utilization of node N_i
C^i	color recorded list of Node N_i
B^n_{is}	bandwidth utilization between nodes N_i and N_s
B^o_{jh}	bandwidth utilization between operators o_j and o_h
l^n_i	load of node N_i
l^o_j	load of operator o_j
$O_{j(j+1)}$	super-operator combining o_j and o_{j+1}
$l^o_{j(j+1)}$	load of super-operator $O_{j(j+1)}$
C^b_{is}	available bandwidth capacity between nodes N_i and N_s
C^c_i	available CPU capacity of node N_i
N^c_i	total CPU/processing capacity of node N_i
B^c_{is}	total bandwidth capacity between nodes N_i and N_s
c_j	cost of operator o_j
s_{kj}	selectivity for the k^{th} input stream to down operator o_j
A	operator allocation matrix $\{a_{ij}\}_{n\times m}$
B^n	node connection used for the stream processing matrix $\{b^n_{is}\}_{n\times n}$
B^o	operator connection matrix $\{b^o_{jh}\}_{m\times m}$
Z	node utilization ratio
F	average selected-nodes' CPU utilization ratio
$B^o_{[j(j+1)][h(h+1)]}$	bandwidth utilization between super-operators $O_{j(j+1)}$ and $O_{h(h+1)}$
C_s	summary color recorded list of non-peer operator
S_{is}	stream between nodes N_i and N_s

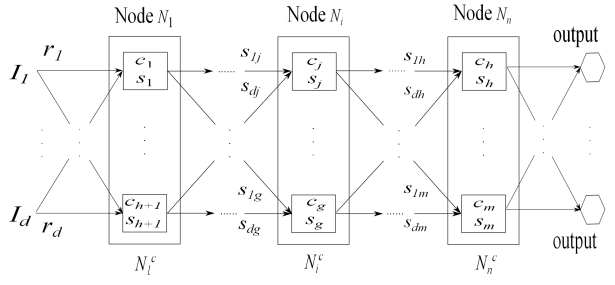


Fig. 1 Example of query diagram.

function of selectivities and the system input stream rates.

Our economical operator distribution algorithm is based on a BP model where the load of each operator o_j for input stream rate r_k can be written as a linear function, i.e.,

$$l_j^o = \Sigma_{k=1}^{d} r_k s_{kj} c_j, \quad 1 \le j \le m, \tag{1}$$

and the load of each node N_i for r_k can be written as a linear function, i.e.,

$$l_i^n = \Sigma_{j=1}^{m} \Sigma_{k=1}^{d} r_k s_{kj} c_j a_{ij}, \quad 1 \le i \le n, \tag{2}$$

and the bandwidth utilization between two operators, o_j and o_h, for r_k can be written as a linear function, i.e.,

$$B_{jh}^o = \Sigma_{k=1}^{d} r_k s_{kj} b_{jh}^o, \quad 1 \le j < h \le m, \tag{3}$$

and the bandwidth utilization between two nodes, N_i and N_s that install operators o_j and o_h, respectively, for r_k can be written as a linear function, i.e.,

$$B_{is}^n = \Sigma_{j=1}^{m-1} \Sigma_{h=j+1}^{m} \Sigma_{k=1}^{d} r_k s_{kj} a_{ij} b_{jh}^o a_{sh}, \quad 1 \le i < s \le n. \tag{4}$$

Through the above formulas, we can deduce the following equations:

$$l_i^n = \Sigma_{j=1}^{m} l_j^o a_{ij}, \quad 1 \le i \le n, \tag{5}$$

$$B_{is}^n = \Sigma_{j=1}^{m-1} \Sigma_{h=j+1}^{m} B_{jh}^o b_{is}^n, \quad 1 \le i < s \le n. \tag{6}$$

For simplicity, we assume that the system input stream rates are variables while the operator costs and selectivities are constants. Because the cost and the selectivity of an operator are related to the attribution of the operator, we can get the values of them through test in advance. Under this assumption, all operator loads are the linear functions of the system's input stream rate.

2.4 Mathematical Model of Economical Operator Distribution Plan

Our goal is to find an energy management scheme that stars with a system using the minimum number of nodes, for decreasing the network resources waste. We aim to fully utilize the network resources through the EOD plan. We formally state the corresponding EOD model as a bin packing optimization problem as follows:

Economical operator distribution (EOD) problem: Given a sequence $O = (o_1, \cdots, o_m)$ of operators, each with a load l_j^o, a sequence $N=(N_1, \cdots, N_n)$ of nodes, each with an available CPU capacity C_i^c, and a sequence $S = (S_{12}, \cdots, S_{is}, \cdots, S_{(n-1)n}), 1 \le i < s \le n$ of streams, each with an available bandwidth capacity C_{is}^b, we aim to put those operators onto a minimum number of nodes, i.e., find

$$min \quad Z = \frac{\Sigma_{i=1}^{n} n_i}{n}, \quad or \tag{7}$$

$$max \quad F = \frac{\Sigma_{i=1}^{n} U_i^c}{\Sigma_{i=1}^{n} n_i} \tag{8}$$

$$s.t. \ \ l_i^n \le C_i^c \tag{9}$$

$$B_{is}^n \le C_{is}^b \tag{10}$$

$$n_i = \begin{cases} 1, & \text{if and only if } \Sigma_{j=1}^{m} a_{ij} > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{11}$$

Z and F represent the node utilization ratio and the average selected-nodes' CPU utilization ratio, respectively. $\Sigma_{i=1}^{n} n_i$ represents the number of selected-nodes, and the CPU utilization of the selected node N_i for r_k can be written as a linear function, i.e.,

$$U_i^c = l_i^n. \tag{12}$$

A set of linear constraints on the load of nodes is defined as the load constraints by (9), and a set of linear constraints on the bandwidth among nodes is defined as the bandwidth constraints by (10).

3. Coloring Super-Operator Scheme

In DSPSs, the replication mechanism allows continuous access to data when a node fails. Here we want to exploit the replication mechanism for related but different purposes: for allowing continuous data access not only when a node fails, but also when some nodes are overloaded, i.e., for leveraging a replication-based scheme to safely and easily load shedding some nodes while keeping the QoS guarantees of the system. We define an overloaded node as when a node receives bytes in unit of time beyond the node processing capacity.

When we look at the load shedding [13]–[15], commonly used techniques in load balancing, we find that although they try to maximize the total weighted throughput, they could not keep the QoS guarantees of the system due to dropping some tuples. For instance, if one node is overloaded, it will drop some tuples selectively. In our approach, on the other hand, we make full use of multi-degree operator replicas framework to solve this problem other than the previous methods (illustrated in Sect. 4).

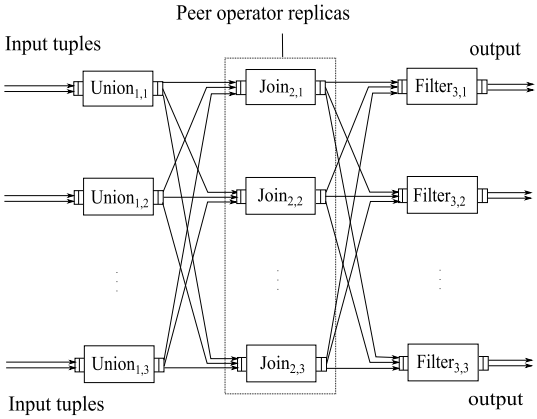


Fig. 2 A framework of multi-degree operator replicas.

3.1 Super Operator Based on Multi-Degree Operator Replicas

As described in Fig. 2, there is a framework of multi-degree operator replicas, such as multi-*Union* operator replicas, multi-*Join* operator replicas and multi-*Filter* operator replicas in which $Union_{1,1}$, $Union_{1,2}$ and $Union_{1,3}$ are each other's replicas, namely, they are peer operator replicas and have the same function in the system. In Fig. 2, each operator runs in parallel and independently. In other words, each operator can receive input tuples in parallel from each upstream operator and send output tuples in parallel to each downstream operator. As shown in Fig. 2, $Join_{2,1}$ can receive input tuples in parallel from $Union_{1,1}$, $Union_{1,2}$ or $Union_{1,3}$. After processing those data, $Join_{2,1}$ sends output tuples in parallel to $Filter_{3,1}$, $Filter_{3,2}$ and $Filter_{3,3}$. Therefore, such a DSPS based on multi-degree operator replicas framework has a good fault-tolerant characteristic. Even in the case of a single operator failure, this DSPS works normally. For example, if $Join_{2,2}$ is broken, downstream operators $Filter_{3,1}$, $Filter_{3,2}$ and $Filter_{3,3}$ can receive data from upstream operators $Join_{2,1}$ or $Join_{2,3}$ (here, we assume one of $Join_{2,2}$'s replicas, $Join_{2,1}$ or $Join_{2,3}$ is assigned to a different node at least).

As explained before, we exploit the replication mechanism for dealing with not only the node failure, but also the node overload. We devise the super-operator mainly for the following reasons, one is to prepare for coloring super-operator and load shedding to keep consecutive and highly-available data stream processing in the case of node failure or input rate bursts. The other is for keeping the SO with "high impact" in processing early. Because [11] argued that assigning the "high impact" operators late may cause the system to significantly deviate from the optimal results, the load balancing of multi-degree operator replicas for initializing an operator distribution plan takes an essential effect in the performance of the system.

We assume there is a coordinator in the system and the loads information of all operators are reported to the coordinator that works as the same as Vivaldi [18]. After statistics collection, the coordinator orders all operators in descending by their loads.

Table 2 An example of the SOs based on multi-degree operator replicas.

Node		N_1	N_2	$\cdots$	N_{m-1}	N_m
RD_2	**the 1st replica:**	o_1	o_2	$\cdots$	o_{m-1}	o_m
	the 2nd replica:	o_2'	o_3'	$\cdots$	o_m'	o_1'
SO:		O_{12}	O_{23}	$\cdots$	$O_{(m-1)m}$	O_{m1}
RD_3	**the 1st replica:**	o_1	o_2	$\cdots$	o_{m-1}	o_m
	the 2nd replica:	o_2'	o_3'	$\cdots$	o_m'	o_1'
	the 3rd replica:	o_3'	o_4'	$\cdots$	o_1'	o_2'
SO:		O_{123}	O_{234}	$\cdots$	$O_{(m-1)m1}$	O_{m12}
			$\cdots$			
RD_r	**the 1st replica:**	o_1	o_2	$\cdots$	o_{m-1}	o_m
	the 2nd replica:	o_2'	o_3'	$\cdots$	o_m'	o_1'
				$\cdots$		
	the r^{th} replica:	o_r'	o_{r+1}'	$\cdots$	o_{r-2}'	o_{r-1}'
SO:		$O_{12\cdots r}$	$O_{23\cdots(r+1)}$	$\cdots$	$O_{(m-1)m\cdots(r-2)}$	$O_{m1\cdots(r-1)}$

As shown in Table 2, there are different replication degrees (RDs) and we can chose the RD according to different application requirements. We denote $RD_x(2 \le x \le r)$ as the replication degree in which r represents the maximum number of operator replicas. We order the operator replicas in each degree, respectively. Therefore, RD_x has x ordered groups. For example, in the case of two-degree operator replicas, there are two ordered groups: one is for the 1^{st} degree, $o_1, o_2, \ldots, o_m$, and the other is for the 2^{nd} degree, $o_1', o_2', \ldots, o_m'$ in which operator o_1' is operator o_1's replica, i.e., they are peer operator replicas and have the same function in the system. In our approach, we sort operators in descending order based on their weight. In the following, we assume that the weight of each operator o_j is equal to its load l_j^o (here, we assume o_1 has the largest weight, o_2 has the second largest weight, and so on). Then we pair the j^{th} operator in the 1^{st} degree replica's ordered list with the $(j+1)^{th}$ operator in the 2^{nd} degree replica's ordered list (for $j = m$, this operator in the 1^{st} degree replica's ordered list will be paired with the first operator in the 2^{nd} degree replica's ordered list as o_{m1} as shown in Table 2). For simplicity, we assume two-degree replication in the following without losing generality. In the case of two-degree replication as shown in Table 2, operator o_1 in the 1^{st} degree of replica's ordered list has the largest weight and operator o_2' in the 2^{nd} degree of replica's ordered list has the second largest weight, then we combine operator o_1 and operator o_2' as a super-operator O_{12}.

Consequently, the load of a super-operator $O_{j(j+1)}$ for input rate r_k can be represented as a linear function, i.e.,

$$l_{j(j+1)}^o = l_j^o + l_{j+1}^o. \tag{13}$$

The bandwidth utilization between super-operators

$O_{j(j+1)}$ and $O_{h(h+1)}$ for input rate r_k can be written as a linear function, i.e.,

$$B^o_{[j(j+1)][h(h+1)]} = B^o_{jh} + B^o_{(j+1)h} + B^o_{j(h+1)} + B^o_{(j+1)(h+1)} = \Sigma^d_{k=1} r_k \times\{s_{kj}(b^o_{(j+1)h} + b^o_{(j+1)(h+1)}) + s_{k(j+1)}(b^o_{jh} + b^o_{j(h+1)})\}. \quad (14)$$

The bandwidth utilization between two nodes N_i and N_s for super-operators can be written as a linear function, i.e.,

$$B^n_{is} = \Sigma^{m-1}_{j=1}\Sigma^m_{h=j+1} B^o_{[j(j+1)][h(h+1)]} b^n_{is}, \quad 1 \le i < s \le n. \quad (15)$$

3.2 Mathematical Model of Coloring Super-Operator

We introduce coloring SO in this paper for the following two reasons: one is for putting the SO with "high-weight" on different nodes, and the other is for overcoming the problems of node failure. If all peer operator replicas are assigned to the same node, the system cannot handle with the problems of node failures. Therefore, we color super-operator based on a coloring bin-packing (CBP) model that assigns peer operator replicas to different nodes, e.g., operator o_2 is assigned on node N_2 while operator $o_2^{'}$ is assigned on node N_1 as shown in Table 2. In other words, we combine one operator with its non-peer operator's replicas as the SO, e.g., operator o_2 is the replica of operator $o_2^{'}$'s peer operator; operator o_1 is o_2's or $o_2^{'}$'s non-peer operator replica. We formally state the coloring super-operator (CSO) problem based on a CBP model.

Coloring super-operator (CSO) problem: Given a sequence $O = (O_{12}, O_{23}, \cdots, O_{m1})$ of SOs and a sequence $N = (N_1, N_2, \cdots, N_n)$ of nodes, each with a color recorded list C^i $(1 \le i \le n)$, we aim to color each SO and put those SOs on appropriate nodes. The concrete steps are as follows:

1. We first color the SOs as $(O_{12}(a_{i1}a_{i2}), O_{23}(a_{i2}a_{i3}), \cdots, O_{j(j+1)}(a_{ij}a_{i(j+1)}), \cdots, O_{(m-1)m}(a_{i(m-1)}a_{im}), O_{m1}(a_{im}a_{i1}))$.

2. $i \leftarrow 1$. We try to assign a candidate SO, $O_{j(j+1)}$, to node $N_i (1 \le i \le n)$.

3. When we try to distribute a candidate SO on node N_i, we first check the value of N_i's color recorded list according to the color of candidate SO, $O_{j(j+1)}(a_{ij}a_{i(j+1)})$, i.e., find the values of a_{ij} and $a_{i(j+1)}$ in the color recorded list C^i:

$$C^i = (a_{i1},\ a_{i2},\ \cdots,\ a_{im}) \quad (16)$$

$$s.t.\ \ a_{ij} = \begin{cases} 1, & \text{one of } o_j\text{'s peer replica is assigned to } N_i \\ 0, & \text{otherwise} \end{cases},$$

$$\Sigma^n_{i=1} a_{ij} = 2. \quad (17)$$

4. If $a_{ij} = 0$ and $a_{i(j+1)} = 0$, it means there are no peer operator's replicas for operators o_j and o_{j+1}, respectively, that are assigned in node N_i. Then, we can put this candidate $O_{j(j+1)}$ in node N_i. After that, the values of both a_{ij} and $a_{i(j+1)}$ are recorded as 1, respectively, in N_i's color recorded list.

5. If $a_{ij} = 1$ or $a_{i(j+1)} = 1$, $i \leftarrow i + 1$ and go to step 3.

For example, if $O_{(j-1)j}$ and $O_{(j+1)(j+2)}$ have been assigned on N_i, the values of $a_{i(j-1)}$, a_{ij}, $a_{i(j+1)}$ and $a_{i(j+2)}$ have been recorded as 1 in N_i's color recorded list, respectively. For candidate $O_{j(j+1)}$, it means its peer operator replicas o_j and o_{j+1} have been assigned in N_i before. Hence, it is prohibitive for the candidate $O_{j(j+1)}$ to be assigned in this node.

4. Load Shedding Method Based on the SO Scheme and the CBP Model

As described in Sect. 2, the EOD plan tends to use a "first-fit" BP model for distributing operators in order to reduce the network resources waste. However, using this scheme, the system cannot tolerate an input rate burst or a spike in an input rate. Here, we take advantage of a simple load shedding scheme based on the SO scheme and the CBP model to solve those problems, namely, if the loads of nodes are over their load level threshold, we alternatively control the non-peer operator replicas to drop processing some tuples in order to alleviate the burden of the overload nodes. In our replication mechanism, we considered the replicas for each query. It means there is no trouble for different users whose queries are processed by different replicas. Consequently, while we drop processing some tuples, we can provide the QoS guarantees of the system for different users.

Load shedding problem: Given a sequence $O = (o_1, \cdots, o_j, \cdots, o_m)$ of operators, a sequence $C = (C^1, \cdots, C^i, \cdots, C^n)$ of nodes' color recorded lists and a summary color recorded list of non-peer operator $C_s = (a_1, \cdots, a_j, \cdots, a_m)$ that is controlled by the coordinator in the system. We aim to utilize a load shedding scheme among non-peer operator replicas to alleviate the burden of the overloaded nodes while keeping the QoS guarantees of the system. The process consists of the following steps:

1. When node N_i is overloaded, we first check the values in C_s, according to node N_i's color recorded list C^i where the values are equal to 1, i.e., find the values about existent operators $(a_{ij} = 1)$ of N_i in C_s:

$$C_s = (a_1,\ a_2,\ \cdots,\ a_m) \quad (18)$$

$$a_j = \begin{cases} 1, & \text{one of } o_j\text{'s peer replica is used for load shedding} \\ 0, & \text{otherwise} \end{cases}, \quad (19)$$

$$s.t.\ \ a_{ij} = \begin{cases} 1, & \text{one of } o_j\text{'s peer replica is assigned to } N_i \\ 0, & \text{otherwise} \end{cases},$$

$$\Sigma^n_{i=1} a_{ij} = 2. \quad (20)$$

2. If $a_j = 0$, then no peer operator replica is used for load shedding for this operator, we can select this operator to give up processing some tuples to alleviate the burden of

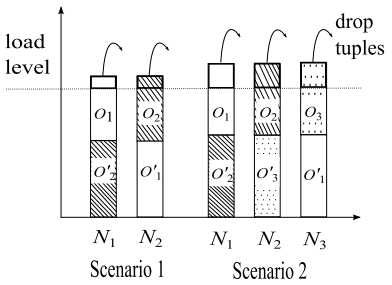


Fig. 3 Load shedding scenarios.

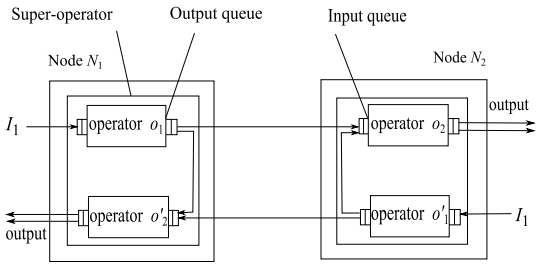


Fig. 4 Stream system based on EFTLBS.

this node.

3. If $a_j = 1$, it means its peer replica is used for load shedding, it is prohibitive for this operator to be used for load shedding.

Figure 3 shows examples of load shedding. In this figure, scenarios 1 and 2 are based on the query graph of Fig. 4 in which peer operator replicas run in parallel and independently so that each downstream operator can use whichever data arrives first. We assume that each operator has the load that is expressed by different patterns rectangle as shown in Fig. 3. If any load of nodes is over their load level threshold, we drop processing some tuples to alleviate the burden of those nodes. In the case of scenario 1, we assume nodes N_1 and N_2 are overloaded successively. For node N_1, we first check the value of operators o_1 and $o_2^{'}$ in the summary color list of non-peer operator. If any one of o_1 or $o_2^{'}$'s value is equal to 0, we can randomly select one operator to drop processing some tuples until N_1's load is under the load level threshold (assume selecting operator o_1 in the figure). Then, the value of operator o_1 in the summary color list of non-peer operator must be recorded as 1. After that, for node N_2, we also first check the value of operators o_2 and $o_1^{'}$ in the summary color list of non-peer operator. Because operator o_1 has been selected for load shedding and the value of operator o_1 in the summary color list of non-peer operator has been equal to 1, we just can select operator o_2 to drop processing some tuples until N_2's load is under the load level threshold. In such scheme, although we drop some part of tuples, we also can get good quality query results by one of chains, from operator $o_1^{'}$ to operator $o_2^{'}$ in the figure, thereby keeping the high QoS guarantees of the system. In the case of scenario 2, we also assume nodes N_1, N_2 and N_3 are overloaded successively. We control operator o_1, operator o_2 and operator o_3 to drop processing some tuples, respectively, until those loads of nodes are under the load level threshold. Although we drop processing some tuples for load balancing of the data stream system, we can get good query results by one of chains, from operator $o_1^{'}$ to operator $o_2^{'}$ to operator $o_3^{'}$, thereby keeping the high QoS guarantees of the system. If all operator replicas are already used for load shedding, we propose to select the operator that has the most load for load shedding in this scheme. On the other hand, we active this operator's backup that exits in the non-overloaded node to keep the QoS guarantees of the system that works as the same as paper [19].

5. Economical and Fault-Tolerant Load Balancing Strategy

Through seamlessly combining the EOD plan discussed in Sect. 2, the CSO scheme discussed in Sect. 3, and the load shedding method discussed in Sect. 4, we present here an economical and fault-tolerant load balancing strategy (EFTLBS). The EFTLBS algorithm consists of two parts as shown in Algorithm 1. One is for the operator distribution at the beginning of the system running. The other is for controlling overloaded nodes while the system runs. For simplicity, we assume that the degree of operator replicas is two in the following without losing generality.

The steps of controlling overloaded nodes have been explained in Sect. 4. The steps of the operator distribution are as follows:

Phase 1: Ordering Operators and Nodes. This phase sorts each degree's operator replicas in descending order based on the load of operators, l_j^o (line 8). Similar sorting orders are commonly used in greedy load balancing and bin packing algorithms [11], [20]–[22]. We also sort nodes in descending order based on their available capacity of nodes (line 9) where the details have been explained in Sects. 3.1 and 3.2. Needless to say, we do not need to take this step into consideration in such an environment that all nodes have the same available capacity.

Phase 2: Combining and Coloring SOs. In this phase, we pair the operator having the largest weight with the operator having the second largest weight, pair the operator having the second largest weight with the operator having the third largest weight as super-operator, and so on (line 11). For solving the problems of server failures and input rate bursts, we color those super-operators as $O_{j(j+1)}(a_{ij}a_{i(j+1)})$ (line 12).

Phase 3: Allocating SOs by the EOD plan. This phase goes through the ordered list of super-operators and iteratively assigns them to those candidates. Our basic destination node selection policy is based on a "first-fit" BP model: at each step, we must judge whether the load, $l_{j(j+1)}^o$, of a candidate SO, is less than the available capacity of node, C_i^c. If it does not meet the above mentioned condition, we will try to assign this SO to node N_{i+1} (line 27). After that, we will check the value of a_{ij} and $a_{i(j+1)}$ in N_i's color recorded list, C^i (lines 14-16). If a_{ij} and $a_{i(j+1)}$ are unequal to 1, it means any one of this SO's peer replicas

Algorithm 1 The EFTLBS algorithm

Pseudocode 1: Operator distribution before system running

```
1  //Initialization
2  for i = 1, ..., n − 1; for s = i + 1, ..., n
3     for j = 1, ..., m − 1; for h = j + 1, ..., m
4        a_ij ← 0, b^n_is ← 0, b^o_jh ← 0
5  for i = 1, ..., n; for j = 1, ..., m; l^n_i ← 0, C^i ← 0
6  for j = 1, ..., m; for k = 1, ..., d; l^o_j ← sqrt(l^o_j1 ^2 + ··· + l^o_jk ^2)
7  //Ordering Operators and Nodes
8  sort operators by l^o_j in descending
9  sort nodes by C^c_i in descending
10 //Combining and Coloring Super-Operators
11 combine the j^th in one replica's order list with the (j + 1)^th in
   another replica's order list as SO: O_j(j+1)
12 color O_j(j+1) as O_j(j+1)(a_ij a_i(j+1))
13 //Allocating Super-Operators by the EOD plan
14 for j = 1, ··· , m //classify operators
15    for i = 1, ··· , n //classify nodes
16       if l^o_j(j+1) ≤ C^c_i and (a_ij ≠ 1 and a_i(j+1) ≠ 1) in color recorded
         list C^i
17          if b^n_is ≠ 1
18             assign O_j(j+1) to node N_i, i.e., a_ij ← 1,
               a_i(j+1) ← 1
19             break //finish assigning O_j(j+1)
20          else [b^n_is = 1 and (b^o_jh = 1 or b^o_j(h+1) = 1 or b^o_(j+1)h = 1
            or b^o_(j+1)(h+1) = 1)]
21             if bandwidth B^o_[j(j+1)][h(h+1)] ≥ C^b_is //assume N_i is
               connected to N_s and O_j(j+1) is connected to O_h(h+1)
22                try to assign this candidate SO to the next node N_i+1
23             else
24                assign O_j(j+1) to node N_i, i.e., a_ij ← 1,
                  a_i(j+1) ← 1
25                break //finish assigning O_j(j+1)
26       else
27          try to assign this candidate SO to the next node N_i+1
28 //Assume N_i is the selected node and assign O_j(j+1) to N_i
29 a_sj ← 1;
30 reset l^n_s ← l^n_s + l^o_j + l^o_j+1, C^c_s ← N^c_s − l^n_s
31 //Assume N_i is connected to N_s and O_j(j+1) is connected to O_h(h+1)
32 b^n_is ← 1;
33 for i = 1, ..., n − 1; for s = i + 1, ..., n
34    for j = 1, ..., m − 1; for h = j + 1, ..., m
35       B^n_is ← B^n_is + B^o_[j(j+1)][h(h+1)], C^b_is ← B^c_is − B^n_is
```

Pseudocode 2: Controlling overload nodes while system running

```
1  //Assume there is at least one operator that is allocated in node N_i
2  //when node N_i is overloaded do
3     for j = 1, ··· , m //classify operators
4        //check the values of a_ij in C^i
5        if a_ij = 1
6           //check the values of a_j in C_s
7           if a_j = 1
8              it's prohibitive for this operator to be used for load shedding
9           else
10             select this operator to give up processing some tuples
11             break //finish selecting an operator for load shedding
```

has not been allocated in N_i. Then, we continuously check the value of b^n_{is} which represents the connection used for the stream processing between nodes N_i and N_s.

If b^n_{is} is unequal to 1, it means the connection between nodes N_i and N_s is not used for the stream processing, we can put $O_{j(j+1)}$ into this node N_i (lines 17-19). Otherwise, we must continuously compare the bandwidth utilization between nodes N_i and N_s with the available bandwidth capacity between nodes N_i and N_s (lines 20-24) in which we should check that which operators in super-operators are connected with each other (line 20). We assume there exists a connection between Super-operators (Assume N_i is connected to N_s and $O_{j(j+1)}$ is connected to $O_{h(h+1)}$), then we have to check if the bandwidth utilization of this stream is greater than the available capacity of bandwidth between nodes N_i and N_s (line 21). If meeting the above condition, try to assign this SO to node N_{i+1} for this SO (line 22). Then, we repeat the above steps until this candidate SO is assigned onto an appropriate node. If the bandwidth utilization of this stream is less than the available capacity of bandwidth between nodes N_i and N_s, we can put $O_{j(j+1)}$ onto this node N_i (lines 23-24). Then we reset the load of a node, the bandwidth utilization of a stream, the available CPU capacity of a node and the available bandwidth capacity of a stream for the next candidate SO assignment (lines 28-35).

6. Experimental Evaluation

6.1 Setup

The experiments run on a machine with an Intel®Core™ duo CPU 3.33 GHz and 4.00 GB main memory underlying Ubuntu 9.10. We built simulations using ns-3 [23]. To send tuples and invoke remote procedures, we used TCP sockets. We used a pseudo stream operator which had a constant processing cost per tuple and was a filtering operator with 50% selectivity, and the computing powers of all nodes were identical for simplicity. Our experiments, except those in Sects. 6.6 and 6.7, used a settled query graph as shown in Fig. 5 for simplicity without losing generality [13], [16], [24]. In Fig. 5, we used six operators, namely, three operators and their replicas, and we provided five nodes. In the future, we would like to discuss for large number of nodes. For the experiment in Sect. 6.6, we set the total input rate of the system was the same and constant, and the ratio of total input rate of the system over the average node processing capacity was 0.5. For the experiment in Sect. 6.7, we used the

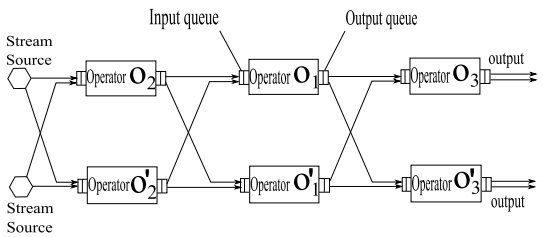


Fig. 5 An example of query graph based on two-degree of replication.

query graph shown in Fig. 5, but the number of replication's degree varied. Given the query graph, we deployed those operator replicas based on the loads information of operators and nodes under different operator distribution plans in which the load information was obtained statistically from the coordinator. In the following evaluations, the node utilization ratio defined by (7) and the average selected-nodes' CPU utilization ratio defined by (8) were used.

6.2 Operator Distribution Plans

We compared our method, EFTLBS, with two alternative distribution approaches. One, called Resilient Operator Distribution (ROD) load balancing that was designed in [7], orders operators by their loads and assigns operators in descending order to the currently least loaded node. It assigns operators in the following steps: (1) Sort operators in descending order. (2) Assign the most loaded candidate operator to the currently least loaded node. (3) Repeat steps (1) and (2) until all operators are assigned. The other algorithm, called economical operator distribution (EOD), prefers to put operators that are sorted by their loads in descending order onto the nodes that are sorted by their available CPU capacities in descending order based on a "first-fit" bin-packing model to minimize energy consumption. It assigns operators in the following steps: (1) Sort nodes and operators in descending order, respectively. (2) Try to assign the most loaded candidate operator to the least loaded node, say N_s. (3) Assign the following candidate operators to N_s until this node has no capacity to install a new candidate operator. (4) If node N_s is full, then try to assign the following operators to a new node N_{s+1} instead of N_s and repeat steps (3) and (4) until all operators are assigned. We assumed there existed at least one node that had enough CPU capacity for each operator.

6.3 Comparing the Techniques based on Bandwidth Limitation

We compared the reliability of EFTLBS, ROD and EOD under the limitations of each stream bandwidth as well as each computer's CPU capacity. We simulated this experiment in 1 minute and simulated a spike in an input rate at time 20 sec and a failure of one node at time 40 sec in the duration. The latency of a tuple L is defined as the difference between the time that tuple a was produced by the upstream operator o_2 or o_2' and the time that tuple a was received by the downstream operator o_3 or o_3'. We define the increasing delay L_B that is caused by the bandwidth overload in the ROD and EOD approaches as $\frac{1}{\lambda}\left(\frac{B^n_{is(tuples/sec)}}{B^c_{is(tuples/sec)}} - 1\right)$, where $\frac{B^n_{is}}{B^c_{is}}$ is greater than 1, λ is equal to r_k, B^n_{is} expresses the bandwidth utilization between nodes N_i and N_s, and B^c_{is} represents the total bandwidth capacity between nodes N_i and N_s.

As shown in Fig. 6 (a), it is obvious that the performance of EFTLBS is significantly better than the average performance of the other techniques because it not only assigns peer replicas to different nodes, but also tries to take the bandwidth limitation into consideration. Therefore, it could provide low latency from the beginning of the stream processing. Even facing a spike in an input rate at 20 sec, it could use a load shedding method to alleviate the burden of the stream bandwidth to keep stable latency stream processing. Moreover, because of using CSO, it could tolerate the node failure's at 40 sec and get good quality query results from another smooth query chain. The EOD technique fares the worst, on the contrary, because it tries to keep all nodes on the same node that results in high-density connection between two nodes so that it experiences a higher latency from the beginning of stream processing. On the other hand, it does not regard bandwidth limitation among nodes so that, when facing a spike in an input rate at 20 sec, the stream between two nodes is too overloaded that results in high-latency to 1.25 sec. In addition, in the case of node failure at 40 sec, o_3 and o_3' did not produce any output because it no longer received tuples from o_1 or o_1' that were assigned in the same broken node. ROD, which dose not consider the bandwidth limitation or the system's fault-tolerance, could not put up with a spike in an input rate at 20 sec or the node's failure at 40 sec. Figure 6 (b) expresses the relation in terms of L, λ, B^n_{is}, B^c_{is}, and the average processing time per input tuple, c, for the ROD and EOD methods. As shown in Fig. 6 (b), based on the setting that λ, B^c_{is} and c are constants, L is linear with B^n_{is}, ($L_B < \frac{1}{\lambda}$). As the bandwidth is overloaded, L is in proportion to B^n_{is}. Note that the EFTLBS could give low latency.

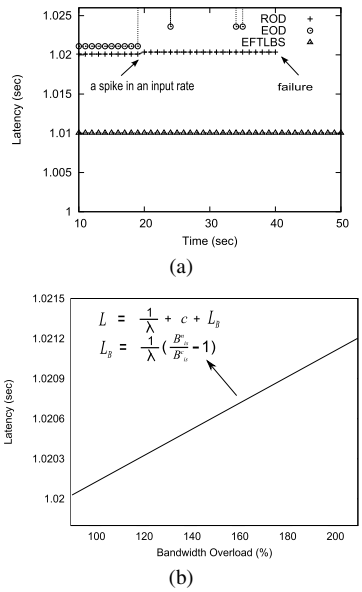


Fig. 6 Comparing the techniques based on bandwidth limitation.

6.4 Varying the Numbers of Input Streams

We varied the number of input streams from 2, to 4, 6, 8, and 10 in which each input stream had the same input rate. We set the ratio of each input rate of the system over the av-

erage node processing capacity as 0.4. As shown in Fig. 7, it is obvious that the performance of ROD was significantly worse than the average performance of the other algorithms as the number of input streams increased, because it tries to keep the largest loaded operators on the least loaded nodes so that the loads of all operators are shared by all nodes, thereby causing waste and insufficient utilization to the resources. On the contrary, EOD always tries to keep operators on the same node for saving energy consumption, and hence the average selected-node CPU utilization ratio of this method is higher than the other methods and the node utilization is lower than the other methods. On the other hand, the EFTLBS uses coloring SO for trying to allocate peer operator replicas to different nodes so that it has more node utilization than the EOD plan.

6.5 Varying the Average Node Processing Capacities

In this experiment, we set the total input rate of the system was the same and constant. We first assigned those operators through the coordinator by using different distribution placement methods. Then we compared the average selected-node CPU utilization ratio and the node utilization ratio of those algorithms under the variation of the average node processing capacity in which a node processing capacity corresponds to this node's total CPU capacity, N_i^c. As shown in Fig. 8, the performance of ROD was significantly worse than the average performance of the other algorithms as the ratio of the average processing capacity of nodes over the total input rates of the system increased, because the ROD method assigns operators to all nodes so that the load of all operators are shared by all nodes. Consequently, the average node CPU utilization ratio of ROD is lower than the other distribution plans and ROD has the highest node utilization ratio, 1. The average node CPU utilization is also in inverse proportion to the average node processing capacity. The EFTLBS lies between ROD and EOD because, while the EFTLBS tries to save energy consumption, this algorithm assigns operators based on coloring SO and thus tends to allocate peer operator replicas to different nodes. Because EOD takes reducing energy consumption into consideration it tries to put operators on the same node. Therefore, the average node CPU utilization ratio of the EOD method is higher than the ROD and EFTLBS plans and the node utilization ratio of the EOD method is lower than the ROD and EFTLBS plans.

6.6 Varying the Numbers of Operators

In this experiment, for the ROD method, as shown in Fig. 9, the average selected-node CPU utilization is in inverse proportion of the number of operators. The reason is the loads of all operators are shared by all nodes so that the more operators there are, the more nodes will be used. On the other hand, comparing the EFTLBS with the EOD plan, we can see they have almost the same average selected-node CPU utilization and the node utilization. Because the total input rate of the system was the same, the more operators there are, the smaller input rate each operator has. However, because of the impact of SO, the EFTLBS must put peer operator replicas into different nodes so that it uses more nodes than the EOD plan as the number of operators increasing to 10, as shown in Fig. 9.

6.7 Impact of Replication

In this experiment, we set each input rate of the system was the same and constant, and the ratio of each input rates of the system over the average node processing capacity was 0.2. Figure 10 (a) shows, as the increasing of the replication's degree, the average selected-node CPU utilization ratio also increased. Figure 10 (b) shows the variation of node utilization ratio, as the degree of replication increased from 2, to 3, 4, and 5. Because there was enough processing capacity

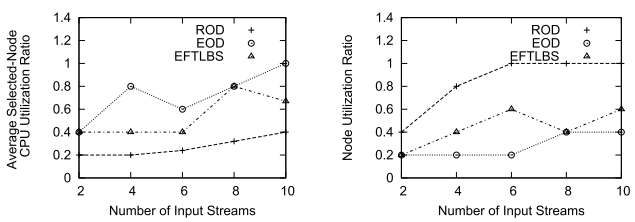


Fig. 7 Impact of the numbers of input streams.

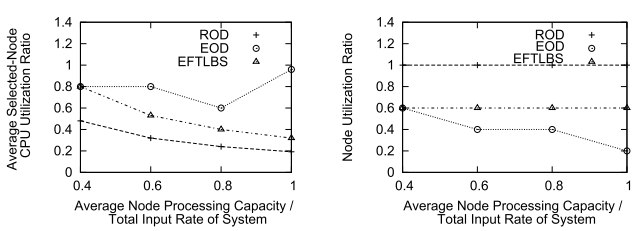


Fig. 8 Impact of the average node processing capacitie.

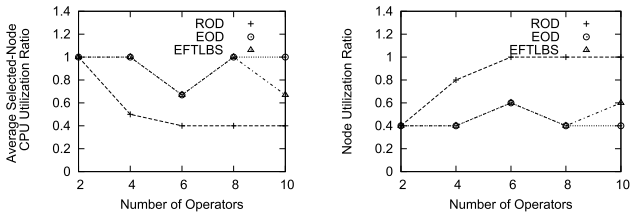


Fig. 9 Impact of the numbers of operators.

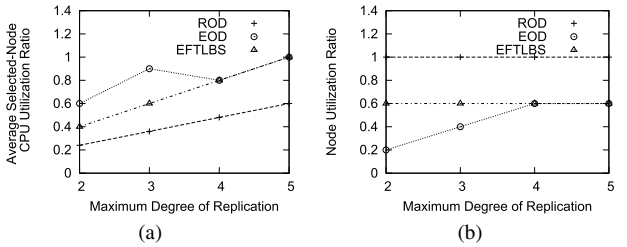


Fig. 10 Impact of degrees of replication.

for each node to process those input rates, we can see as the average selected-node CPU utilization ratio was increasing the node utilization ratio was almost constant.

From the experimental evaluation in Sect. 6.3, we can conclude that as the increasing of the replication's degree, the performance of fault-tolerance of each method is improved. Because even some nodes failed, we can get query results from other good chains. For the performance of load balancing, as we also evaluated in Sect. 6.3, even facing an input rate burst resulting in the overload of a node, the EFTLBS could use a load shedding method to keep stable latency stream processing in any degree of replication. On the contrary, the EOD technique cannot deal with an input rate burst, even as the increasing of the replication's degree. Because it tries to keep all operators on the same node, for this stream processing system, the selected-nodes have no extra CPU to handle with the load balancing of the system. The ROD plan can tolerate an input rate burst, because it tries to allocate operators on the most idle node so that, for this stream processing system, the selected-nodes have extra CPU to handle with the load balancing of the system. We would like to make some extension about the fault-tolerance and efficient load balancing in a real environment in the future.

7. Related Work

Load balancing has been explored extensively for conventional distributed and parallel computing systems [7]–[11], [13]–[15]. In principle, those load balancing schemes are categorized into dynamic load distribution plans, static load distribution plans and load shedding techniques. The dynamic operator allocation techniques are used in papers [7]–[10] in which they migrate the operators for the load balancing of the system when a node is overloaded. Operator movement is too expensive to alleviate short-term bursts; moreover some systems do not support the ability to move operators dynamically. In our method, therefore, we propose a new method to realize load balancing without dynamic operator migration.

Approaches in [7], [11] apply static load distribution techniques where they provide an elastic initial distribution plan to avoid overload for input rates without operator movement. However, they do not regard energy consumption that is a crucial and rising problem in the real world. Hence, we take energy consumption into consideration.

Previous papers [13]–[15] use load shedding techniques to handle with the overload of nodes that drop part of data in the case of the servers overloaded during the data stream processing. Although they try to maximize the total weighted throughput, they could not keep good quality query results due to dropping some tuples. Therefore, we make full use of multi-degree operator replicas framework to solve this problem. In addition, all of those approaches are based on one degree of replication where they do not load balances multi-degree operator replicas and they also assume that the bandwidth is not limited. Therefore, those techniques are unsuited for such environments in which multi-degree operator replicas run in parallel and independently [16].

In summary, to overcome the limitations of the previous techniques, we propose a new approach, the economical and fault-tolerant load balancing strategy (EFTLBS) that based an operator replication mechanism and a load shedding mechanism to realize continuous and highly-available data stream processing without dynamic operator migration over wide area networks.

8. Conclusion

We have argued that the load balancing of continuous and high-available data stream processing applications raises novel challenges and opportunities over wide area networks. We have proposed the economical and fault-tolerant load balancing strategy (EFTLBS), which fully utilizes the network resources to realize continuous and highly-available data stream processing without dynamic operator migration over wide area networks. Through simulations, we have confirmed that our approach is better than the other techniques in economical, continuous and highly-available data stream processing under the constraints of each stream bandwidth as well as each node's CPU capacity. In other words, the performance of the EFTLBS is balanced in terms of the ratios of average selected-node CPU utilization and node utilization.

We proposed a novel load balancing framework that can provide continuous and highly-available data stream processing over wide area networks. In our framework, we use static data collection of the loads information so that such a scheme may cause the deviation of loads information due to networks properties. On the other hand, we set those parameters on the basis of paper [25]. Needless to say, in future work, we plan to study self-tuning techniques for operator distribution and do some experiments in a real environment.

References

[1] D.J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S.B. Zdonik, "The design of the Borealis stream processing engine," Proc. Second Biennial Conference on Innovative Data Systems Research, CIDR, pp.277–289, Jan. 2005. Available at http://www.cidrdb.org/cidr2005/papers/P23.pdf

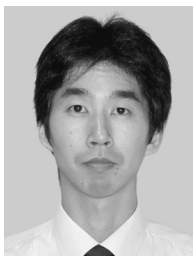
[2] D.J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," The VLDB Journal, vol.12, no.2, pp.120–139, 2003. DOI 10.1007/s00778-003-0095-z.

[3] M. Balazinska, H. Balakrishnan, S.R. Madden, and M. Stonebraker, "Fault-tolerance in the borealis distributed stream processing system," ACM Trans. Database Syst., vol.33, no.1, pp.1–44, 2008. DOI 10.1145/1331904.1331907.

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Continuous dataflow processing for an uncertain world," Proc. First Biennial Conference on

Innovative Data Systems Research, CIDR, Jan. 2003. Available at http://www.cidrdb.org/cidr2003/program/p24.pdf
[5] J.H. Hwang, Y. Xing, U. Çetintemel, and S.B. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," Proc. 23rd International Conference on Data Engineering, ICDE, pp.176–185, 2007. DOI 10.1109/ICDE.2007.367863.
[6] M.A. Shah, J.M. Hellerstein, and E. Brewer, "Highly available, fault-tolerant, parallel dataflows," Proc. 2004 ACM SIGMOD international conference on Management of data, SIGMOD, New York, NY, USA, pp.827–838, ACM, 2004. DOI 10.1145/1007568.1007662.
[7] N. Tatbul, Y. Ahmad, U. Çetintemel, J.H. Hwang, Y. Xing, and S.B. Zdonik, "Load management and high availability in the borealis distributed stream processing engine," in GeoSensor Networks, ed. S. Nittel, A. Labrinidis, and A. Stefanidis, Lecture Notes in Computer Science, vol.4540, pp.66–85, Springer Berlin / Heidelberg, 2008. DOI 10.1007/978-3-540-79996-2_5.
[8] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S.B. Zdonik, "Scalable distributed stream processing," Proc. First Biennial Conference on Innovative Data Systems Research, CIDR, 2003. Available at http://www.cidrdb.org/cidr2003/program/p23.pdf
[9] M. Balazinska, H. Balakrishnan, and M. Stonebraker, "Contract-based load management in federated distributed systems," Proceedings of the First Symposium on Networked Systems Design and Implementation, NSDI, pp.197–210, 2004. http://www.usenix.org/events/nsdi04/tech/balazinska.html
[10] Y. Xing, S. Zdonik, and J.H. Hwang, "Dynamic load distribution in the borealis stream processor," Proceedings of the 21st International Conference on Data Engineering, ICDE, Los Alamitos, CA, USA, pp.791–802, IEEE Computer Society, 2005. DOI 10.1109/ICDE.2005.53.
[11] Y. Xing, J.H. Hwang, U. Çetintemel, and S. Zdonik, "Providing resiliency to load variations in distributed stream processing," Proc. 32nd international conference on Very large data bases, VLDB, pp.775–786, VLDB Endowment, 2006. Available at http://www.vldb.org/conf/2006/p775-xing.pdf
[12] W. Lang, J.M. Patel, and J.F. Naughton, "On energy management, load balancing and replication," SIGMOD Rec., vol.38, no.4, pp.35–42, Dec. 2009. DOI 10.1145/1815948.1815956.
[13] N. Tatbul, U. Çetintemel, and S.B. Zdonik, "Staying FIT: Efficient load shedding techniques for distributed stream processing," Proc. 33rd international conference on Very large data bases, VLDB, pp.159–170, 2007. Available at http://www.vldb.org/conf/2007/papers/research/p159-tatbul.pdf
[14] N. Tatbul, U. Çetintemel, S.B. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," Proc. 29th International Conference on Very Large Data Bases, VLDB, pp.309–320, 2003. Available at http://www.vldb.org/conf/2003/papers/S10P03.pdf
[15] N. Tatbul and S. Zdonik, "Dealing with overload in distributed stream processing systems," Proc. 22nd International Conference on Data Engineering Workshops, ICDE Workshops, Los Alamitos, CA, USA, p.24, IEEE Computer Society, 2006. DOI 10.1109/ICDEW.2006.45.
[16] J.H. Hwang, U. Çetintemel, and S.B. Zdonik, "Fast and highly-available stream processing over wide area networks," Proc. 24th International Conference on Data Engineering, ICDE, pp.804–813, 2008. DOI 10.1109/ICDE.2008.4497489.
[17] M. Castro and B. Liskov, "Practical byzantine fault tolerance," Proc. Third Symposium on Operating Systems Design and Implementation, OSDI, Berkeley, CA, USA, pp.173–186, USENIX Association, 1999. http://portal.acm.org/citation.cfm?id=296806.296824
[18] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," SIGCOMM Comput. Commun. Rev., vol.34, no.4, pp.15–26, Aug. 2004. DOI 10.1145/1030194.1015471.
[19] F. Xiao, K. Nagano, T. Itokawa, T. Kitasuka, and M. Aritsugi, "A self-recovery technique for highly-available stream processing over local area networks," Proc. 2010 IEEE Region 10 Conference, TENCON, pp.2406–2411, Nov. 2010. DOI 10.1109/TENCON.2010.5685904.
[20] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson, Approximation algorithms for bin packing: a survey, pp.46–93, PWS Publishing Co., Boston, MA, USA, 1997. http://portal.acm.org/citation.cfm?id=241938.241940
[21] J. Hromkovic, Algorithmics for hard problems, Springer, 1998.
[22] V.V. Vazirani, Approximation Algorithms, Springer, 2001.
[23] The ns-3 network simulator. http://www.nsnam.org/
[24] J.H. Hwang, U. Çetintemel, and S.B. Zdonik, "Fast and reliable stream processing over wide area networks," Proc. 23rd International Conference on Data Engineering Workshop, ICDEW, pp.604–613, 2007. DOI 10.1109/ICDEW.2007.4401047.
[25] J.H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik, "High-availability algorithms for distributed stream processing," Proc. 21st International Conference on Data Engineering, ICDE, Los Alamitos, CA, USA, pp.779–790, IEEE Computer Society, 2005. DOI 10.1109/ICDE.2005.72.

Fuyuan Xiao received her B.E. degree in computer science and technology from Guilin University of Technology, China, in 2008. She received her M.E. degree in information science and engineering from Guilin University of Technology, China, in 2011. Since 2011, she has been studying for her D.E. degree at the Graduate School of Science and Technology, Kumamoto University, Japan. Her research interests include database systems and parallel/distributed data processing.



Teruaki Kitasuka received his B.E. Degree from Kyoto University, M.E. Degree from Nara Institute of Science and Technology (NAIST), and D.E. Degree from Kyushu University, Japan, in 1993, 1995 and 2006, respectively. He worked for SHARP Corporation from 1995 to 2001. He was a research associate in the Faculty of Information Science and Electrical Engineering, Kyushu University, from 2001 to 2007. Since 2007, he has been an associate professor in the Graduate School of Science and Technology, Kumamoto University. His research interests include location-aware systems, and ubiquitous, mobile and embedded computing. He is a member of IPSJ and IEEE.

Masayoshi Aritsugi received his B.E. and D.E. degrees in computer science and communication engineering from Kyushu University, Japan, in 1991 and 1996, respectively. From 1996 to 2007, he was with the Department of Computer Science, Gunma University, Japan. Since 2007, he has been a Professor at the Graduate School of Science and Technology, Kumamoto University, Japan. His research interests include database systems and parallel/distributed data processing. He is a member of IPSJ, ACM, IEEE-CS, and DBSJ.