Dynamic Load Distribution in the Borealis Stream Processor [*]

Ying Xing
Brown University
yx@cs.brown.edu

Stan Zdonik
Brown University
sbz@cs.brown.edu

Jeong-Hyon Hwang
Brown University
jhhwang@cs.brown.edu

Abstract

Distributed and parallel computing environments are becoming cheap and commonplace. The availability of large numbers of CPU's makes it possible to process more data at higher speeds. Stream-processing systems are also becoming more important, as broad classes of applications require results in real-time.

Since load can vary in unpredictable ways, exploiting the abundant processor cycles requires effective dynamic load distribution techniques. Although load distribution has been extensively studied for the traditional pull-based systems, it has not yet been fully studied in the context of push-based continuous query processing.

In this paper, we present a correlation based load distribution algorithm that aims at avoiding overload and minimizing end-to-end latency by minimizing load variance and maximizing load correlation. While finding the optimal solution for such a problem is NP-hard, our greedy algorithm can find reasonable solutions in polynomial time. We present both a global algorithm for initial load distribution and a pair-wise algorithm for dynamic load migration.

1. Introduction

Stream-based continuous query processing fits a large class of new applications, such as sensor networks, location tracking, network management and financial data analysis. In these systems, data from external sources flows through a network of continuous query operators. Since stream-based applications usually involve large volumes of data and require timely response, they could benefit substantially from the additional horsepower of distributed environments [6].

Borealis [1] is a new distributed stream processing engine that is being developed at Brandeis, Brown, and MIT as a follow on to the Aurora project [2]. Borealis attempts to provide a single infrastructure for distributed stream processing that can span diverse processing elements that can be as small as sensors and as large as servers. As a first step in this direction, we restrict this work to the case of clusters of servers with high-speed interconnections.

In Borealis, as in Aurora, a query network is a collection of operators that are linked together in a dataflow diagram. Our operators extend the relational operators to deal with the ordered and infinite nature of streams. A Borealis query network cannot have loops; however, the output of an operator can branch to multiple downstream operators (result sharing) and can be combined by operators with multiple inputs (e.g., Join, Union).

Query optimization in this setting is to a large extent concerned with mapping the operators in a query network to machines in a distributed environment. As the load changes, this mapping will need to change in order to deal with new hot spots. The process of forming the initial mapping and of dynamically redistributing operators is the topic of this paper.

While load balancing and load sharing have been studied extensively in traditional parallel and distributed systems [11, 16], the load distribution problem has not yet been fully studied in the context of push-based stream processing. Traditional load distribution strategies use total load information in decision making because they are designed for pull-based systems where load fluctuation occurs as different queries are presented to the system. In a push-based system, load fluctuation occurs in the arrival rates of the streams. In this case, even when the average load of a machine (or node) is not very high, a node may experience a temporary load spike and data processing latencies can be significantly affected by the duration of the spike. Thus, to minimize data processing latencies we need an approach that can avoid temporary overload as much as possible.

For instance, consider two operator chains with bursty input data. Let each operator chain contain two identical operators with a selectivity of one. When the average input rates of the two input streams are the same, the average loads of all operators are the same. Now consider two operator mapping plans on two nodes. In the first plan, we put each of the two connected operator chains on the same node (call this the *connected plan*). In the second plan, we place each component of a chain on different nodes (call this the *cut plan*). There is no difference between these two

[*] This work has been supported by the NSF under grants IIS-0086057 and IIS-0325838.

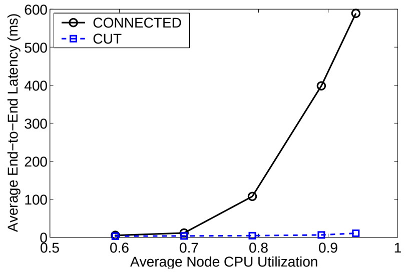

Figure 1: Comparison of different operator mapping plans with fluctuating load

plans from the load balancing point of view. However, suppose the load burst of the two input streams happens at different times, i.e., when the input rate of the first chain is high, the input rate for the second chain is low and vice versa. Then the above two mapping plans can result in very different performance. Figure 1 shows an example performance graph for this kind of workload in which the burst duration and the normal duration are both 5 seconds, and the high (bursty) input rate is twice the low (normal) input rate.

Putting connected operators on different nodes, in this case, achieves much better performance than putting them on the same node (ignoring bandwidth considerations for now). The main difference between these two mapping plans is that since the two input bursts are out of phase, the cut plan ensures that the load variation on each node is very small. In the connected plan, it is much larger. This simple example shows that the average load level is not the only important factor in load distribution. The variation of the load is also a key factor in determining the performance of a push-based system.

In this paper, we propose a new load distribution algorithm that not only balance the average load among the processing nodes, but also minimize the load variance on each node. The latter goal is achieved by exploiting the ways in which the stream rates correlate across the operators. More specifically, we represent operator load as fixed length time series. The correlation of two time series is measured by the correlation coefficient, which is a real number between -1 and 1. Its intuitive meaning is that when two time series have a positive correlation coefficient, then if the value of one time series at certain index is relatively large (in comparison to its mean), the value of the other time series at the same index also tends to be relatively large. On the other hand, if the correlation coefficient is negative, then when the value of one time series is relatively large, the value of the other tends to be relatively small. Our algorithm is inspired by the observation that if the correlation coefficient of the load time series of two operators is small, then putting these operators together on the same node helps in minimizing the load variance.

The intuition of correlation is also the foundation of the other idea in our algorithm: when making operator allocation decisions, we try to maximize the correlation coefficient between the load statistics of different nodes. This is because moving operators will result in temporary poor performance due to the execution suspension of those operators, but if the load time series of two nodes have large correlation coefficient, then their load levels are naturally balanced even when the load changes. By maximizing the average load correlation between all node pairs, we can minimize the number of load migrations needed.

Later, we will see that minimizing the average load variance also helps in maximizing the average load correlation, and vice versa. Thus, the main goal of our load distribution algorithms is to produce a balanced operator mapping plan where the average load variance is minimized or the average node load correlation is maximized. Finding the optimal solution for such a problem requires exhaustive search and is, similar to the graph partitioning problem, NP complete [10]. In this paper, we propose a greedy algorithm that finds a sub-optimal solution in polynomial time. Our experimental results show that the performance of our algorithm is very close to the optimal solution.

In this paper, we present both a global operator mapping algorithm and some pair-wise load redistribution algorithms. The global algorithm is mainly used for initial operator placement. After global distribution, we will use pair-wise algorithms to adapt to load changes. The advantage of using pair-wise algorithms is that it does not require as much load migration as the global algorithm.

The rest of this paper is organized as follows. Section 2 introduces the system model and formalizes the problem. Our algorithms are presented in Section 3. Section 4 analyzes the computation complexity of these algorithms. The experiment results are presented in Section 5. Section 6 discusses related work. Finally, the conclusions and future directions are summarized in Section 7.

2. Problem Description

2.1. System Model and Assumptions

In this paper, we assume a physical architecture of a loosely coupled shared-nothing homogeneous computer cluster. All computers are connected by a high bandwidth network. We assume that the network bandwidth is not a limited resource and network transfer delays as well as the CPU overhead for data stream transfer are negligible [8], [9]. For applications with very high steam rates that may stress the network, connected operators can be encapsulated into super-operators or clusters such that high bandwidth links are internal to a super-operator and thus, do not cross real network links. Operator clustering in the context of fluctuating workload is itself a very challenging topic and is a part of our ongoing work. In this paper, we assume that necessary operator clustering has been done so that we can directly distribute super operators without network bandwidth concern.

In Borealis, most operators (e.g., *Filter*, *Aggregate*, *Join*) provide interfaces that allow them to be moved on the fly. For practical purposes, we consider *SQL-read* and *SQL-*

write boxes to be immovable since their effective state can be huge. When moving a set of operators, their execution is first suspended. Then, the metadata (e.g., operator description and topology) and the operator states (the input queues and the internal operator data structures) are transferred to the receiving node. The receiving node instantiates these operators with the given information and then resumes their execution. In this paper, we assume that all operators with very large states (e.g., databases) are allocated by some other algorithm according to the storage capacities of the nodes. We only consider the mapping and migration of movable operators whose state size is relatively small. Even in this case, the operator migration time is usually much longer than the end-to-end data processing time.

2.2. Load Measurement

In this paper, we consider CPU utilization as the system load. The load of nodes and operators is measured periodically over fixed-length time periods. In each period, the load of an operator is defined as the fraction of the CPU time needed by that operator over the length of the period. In other words, if the average tuple arrival rate in period i for operator o is $\lambda(o)$ and the average tuple processing time of o is $p(o)$, then the load of o in period i is $\lambda(o)\, p(o)$. The load of a node in a given period is defined as the sum of the loads of all its operators in that period.

We define the tuple arrival rate of a stream as the number of tuples that would arrive on the stream when no node in the system is overloaded. If the statistics measurement period is large enough, such "ideal" rates become independent of the scheduling policy. On the other hand, the actual number of tuples that enter each stream per time interval is usually dependent on the scheduling algorithm, especially when some node becomes overloaded.

The ideal tuple arrival rates can be approximately computed from the system input stream rates and the selectivities of the operators. If no global information is available for such computation, an upstream node can tell its downstream nodes the ideal rates of its output streams so that the downstream nodes can compute the ideal rates of their internal data streams locally.

2.3. Statistics Measurement

We measure the load of each operator periodically and only keep the statistics for the most recent k periods. Each statistics measurement period should be long enough so that the measured load is independent of the scheduling policy and any high frequency load fluctuation is smoothed out. The total time of the k statistics measurements is called the *statistics window* of the system. It should be selected large enough to avoid load migration thrashing. The k load values of an operator/node form a load time series for the operator/node.

Given a load time series $S = (s_1, s_2, \ldots, s_k)$, its mean and variance are defined as follows:

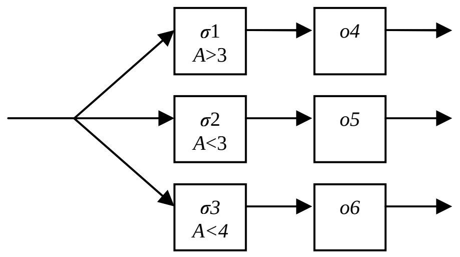

Figure 2: Stream with attribute *A* feeding different filters

$$\mathrm{E}\,S = \frac{1}{k}\sum_{i=1}^{k} s_i$$

$$\mathrm{var}\,S = \frac{1}{k}\sum_{i=1}^{k} s_i^2 - \left(\frac{1}{k}\sum_{i=1}^{k} s_i\right)^2$$

Given two load time series $S_1 = (s_{11}, s_{12}, \ldots, s_{1k})$ and $S_2 = (s_{21}, s_{22}, \ldots s_{2k})$, their covariance $\mathrm{cov}(S_1, S_2)$ and correlation coefficient ρ are defined as follows:

$$\mathrm{cov}(S_1, S_2) = \frac{1}{k}\sum_{i=1}^{k} s_{1i}s_{2i} - \left(\frac{1}{k}\sum_{i=1}^{k} s_{1i}\right)\left(\frac{1}{k}\sum_{i=1}^{k} s_{2i}\right),$$

$$\rho = \frac{Cov(S_1, S_2)}{\sqrt{\mathrm{var}\,S_1}\cdot\sqrt{\mathrm{var}\,S_2}}.$$

In this paper, the variance of the load time series of an operator/node is also called the *load variance* of that operator/node. The correlation coefficient of the load time series of two operators/nodes is also called the *load correlation* of the two operators/nodes. The mean of the load time series of an operator/node is called the *average load* (or simply load) of that operator/node. Load balancing algorithms attempt to balance the average load of the nodes.

Our algorithm is based on the observation that load correlations vary among operators. This variation is a result of more than the fluctuation of different input rates. It also results from the nature of the queries. For example, consider a stream with attribute A feeding different filter operators as depicted in Figure 2. The boxes in the figure represent operators and the arrows represent data streams. It is not difficult to tell that no matter how the input stream rate fluctuates, operators $\sigma 1$, $\sigma 2$ and $\sigma 3$ always have pair-wise load correlation 1, and operators $o4$ and $o5$ always have a load correlation of -1. In addition, operators $o4$ and $o6$ tend to have a negative load correlation, and operators $o5$ and $o6$ tend to have a positive load correlation.

Such query-determined correlations are stable or relatively stable in comparison to input-determined correlations. This feature is important to our algorithms because we use the correlations to determine the locations of the operators. If the correlations are highly volatile, the decisions made may soon loose their effectiveness.

2.4. Optimization Goal

Our goal in load distribution is to minimize the average end-to-end data processing latency. In this paper, we consider two kinds of load distributions: initial operator mapping and dynamic operator redistribution. For the former one, we try to find an operator mapping plan that can minimize the average end-to-end latency. For the latter one, we try to achieve a good balance between the load migration overhead and the quality of the new operator mapping plan.

We have already seen that in a push-based system, minimizing average end-to-end latency can be achieved by minimizing average load variance or maximizing average load correlation. Then our operator mapping problem can be formalized as the follows:

Assume that there are n nodes in the system. Let X_i denote the load time series of node Ni and ρ_{ij} denote the correlation coefficient of X_i and X_j for $1 \le i, j \le n$. We want to find an operator mapping plan with the following properties:

(1) $EX_1 \approx EX_2 \approx \ldots \approx EX_k$

(2) $\frac{1}{n}\sum_{i=1}^{n} \operatorname{var} Xi$ *is minimized or*

(3) $\sum_{1 \le i < j \le n} \rho_{ij}$ *is maximized*

Finding the optimal solution of this problem requires comparison of all possible mapping plans and is NP hard. Thus, our goal is to find a reasonable heuristic solution.

3. Algorithm

3.1. Theoretical Underpinnings

Before discussing our algorithm, it is beneficial to know how to minimize average load variance in the ideal case. In this section, we assume that the total load time series X of the system is fixed, and it can be arbitrarily partitioned across n nodes (this is usually unachievable). We want to find the load partition with minimum average load variance. The result is illustrated by the following theorem:

Theorem 1: *Let the total load of the system be denoted by time series X. Let X_i be the load time series of node i, $1 \le i \le n$, i.e. $X = X_1 + X_2 + \ldots + X_n$. Then among all load balanced mapping plans with* $\mathrm{E}X_1 = \mathrm{E}X_2 \ldots = \mathrm{E}X_n$, *the average load variance*

$$\frac{1}{n}\sum_{i=1}^{n} \operatorname{var} X_i$$

is *minimized if and only if*

$$X_1 = X_2 = \ldots = X_n = \frac{X}{n}.$$

Proof: Let ρ_{ij} be the correlation coefficient between X_i and X_j. Since $X = X_1 + X_2 + \ldots + X_n$, we have

$$\operatorname{var} X = \sum_{i=1}^{n} \operatorname{var} X_i + 2\sum_{1 \le i < j \le n} \rho_{ij}\sqrt{\operatorname{var} X_i}\sqrt{\operatorname{var} X_j} \qquad (1)$$

Since $-1 \le \rho_{ij} \le 1$ and $\sqrt{\operatorname{var} X_i}\sqrt{\operatorname{var} X_j} \le (\operatorname{var} X_i + \operatorname{var} X_j)/2$ we have that

$$\sum_{i=1}^{n} \operatorname{var} X_i \ge \frac{\operatorname{var} X}{n}.$$

The above equality holds if and only if $\rho_{ij}=1$ and $\operatorname{var} X_i = \operatorname{var} X_j$ for all $1 \le i, j \le n$. Using condition $EX_1 = EX_2 = \ldots = EX_n$, we have that $\sum \operatorname{var} X_i$ is minimized if and only if $X_1 = X_2 = \ldots = X_n$. ■

Notice that in the ideal case, when the average load variance of the system is minimized, the average load correlation of the system is also maximized. Naturally, we want to know whether the average load variance is monotonically decreasing with the average load correlation. If so, minimizing average load variance and maximizing average load correlation are then the same. Unfortunately, such a conclusion does not hold in general. It is very easy to find an counter example through simulation. However, in the case of $n = 2$, we can prove that when $\rho_{12} > 0$, the lower bound of the average load variance is a monotone decreasing function of the load correlation coefficient. The conclusion is shown as follows:

Theorem 2: *Given load time series X and X_1, X_2, with $X = X_1 + X_2$, if $\rho_{12} > 0$ then*

$$\frac{\operatorname{var} X}{1+\rho_{12}} \le \operatorname{var} X_1 + \operatorname{var} X_2 \le \operatorname{var} X.$$

The proof is similar to Theorem 1 and is omitted. From this conclusion, we can see that the smaller the correlation coefficient, the larger the lower bound of the average load variance, which means the more room we have for further optimization. Because correlation coefficients are bounded between [-1, 1], it is very easy to use them to check whether a given mapping plan is near optimal and to determine whether redistributing operators between a node pair is necessary. This observation is a very important foundation for one of our optimization techniques.

3.2. Algorithm Overview

In this section, we present a greedy algorithm which not only balances the load of the system, but also tries to minimize the average load variance and maximize the average load correlation of the system.

Our algorithm can be divided into two parts. First, we use a global algorithm to make the initial operator distribution. Then, we switch to a dynamic load redistribution algorithm which moves operators between nodes in a pair-wise fashion. In the global algorithm, we only care about the quality of the resulting mapping plan without considering how much load is moved. In the pair-wise algorithm, we try to find a good tradeoff between the amount of load moved and the quality of the resulting mapping plan.

Both algorithms are based on the basic load-balancing scheme. Thus, if the load of the system does not fluctuate, our algorithm reduces to a load balancing algorithm with a random operator selection policy. When the load of the system fluctuates, we can get load-balanced operator-distribution plans with smaller average load variance and larger average load correlation than the traditional load balancing algorithms.

Since it is easier to understand how to minimize the average load variance between a node pair than among all nodes in the system, we will first discuss the pair-wise algorithm, and then the global algorithm.

3.3. Pair-wise Algorithm

For simplicity, we assume that there is a centralized coordinator in the system and the load information of all nodes is reported periodically to the coordinator. After each statistics collection period, the coordinator orders all nodes by their average load. Then the i^{th} node in the ordered list is paired with the $(n\text{-}i+1)^{th}$ node in the list. In other words, the node with the largest load is paired with the node with the smallest load; the node with the second largest load is paired with the node with the second smallest load, and so on. If the load difference between a node pair is greater than a predefined threshold ε, operators will be moved between the nodes to balance their average load. When necessary, this pair-wise load distribution scheme can be easily extended to a decentralized implementation.

Now, given a selected node pair, we will focus on how to move operators to minimize their average load variance. As we know that there is a tradeoff between the amount of load moved and the quality of the resulting mapping plan, we will first discuss an algorithm that moves the minimum amount of load, and then discuss an algorithm that achieves the best operator mapping quality, and finally, present an algorithm that balances the two goals well.

3.3.1. One-way Correlation Based Load Balancing.

In this algorithm, only the more loaded node is allowed to offload to the less loaded node. Therefore, the load movement overhead is minimized.

Let N_1 denote the more loaded node and N_2 denote the less loaded node. Let the load of N_1 be L_1 and the load of N_2 be L_2. Our greedy algorithm will selects operators from N_1 one by one with total selected load less than $(L_1 - L_2)/2$ until no more operators can be selected. The operator selection policy is inspired by the following observation:

Assume we have only two operators and two nodes. Let the load time series of the operators be S_1 and S_2 respectively and the load correlation coefficient of the two operators be ρ_{12}. Putting the operators on different nodes will results in an average load variance of $(\text{var}S_1 + \text{var}S_2)/2$ and putting the operators on different nodes will results in average load variance of $\text{var}(S_1+S_2)/2$. From the definition of correlation coefficient, we have that

$$\frac{\text{var}(S_1+S_2)}{2} - \frac{\text{var}\,S_1 + \text{var}\,S_2}{2} = \rho_{12}\sqrt{\text{var}\,S_1}\sqrt{\text{var}\,S_2}.$$

Obviously, to minimize average load variance, when $\rho_{12} < 0$, it is better to put the operators together on the same node, and when $\rho_{12} > 0$, it is better to separate them onto different nodes.

Now consider moving operators from N_1 to N_2 following this intuition. Let $\rho(o, N)$ denote the correlation coefficient between the load time series of operator o and the total (sum of) load time series of all operators on N except o. Then from N_1's point of view, it is good to move out an operator that has a large $\rho(o, N_1)$, and from N_2's point of view, it is good to move in an operator that has a small $\rho(o, N_2)$. Considering both nodes together, we prefer to move operators with large $\rho(o, N_1)$ - $\rho(o, N_2)$. Define

$$S(o) = \frac{\rho(o, N_1) - \rho(o, N_2)}{2}$$

as the score of operator o with respect to N_2. Our greedy operator selection policy then selects operators from N_1 one by one with the largest score first.

As the score function in this algorithm is based on the correlation coefficients, and the load can only be moved from one node to the other, this algorithm is called the *one-way correlation-based load balancing algorithm*.

3.3.2 Two-way Correlation-Based Redistribution.

In this algorithm, we redistribute all operators on a given node pair without considering the former locations of the operators. With this freedom, it is possible to achieve the best operator mapping quality.

The operator selection policy in this algorithm is also a score based greedy algorithm. We first start from two “empty” nodes (nodes with non-movable operators only), and then assign movable operators to these nodes one by one. In order to balance the load of the two nodes, for each assignment, we select the less loaded node as the receiver node. Then from all operators that have not been assigned yet, we compute their score with respect to the receiver node and assign the operator with the largest score to that node. This process is repeated until all operators are assigned. Finally, we use the above one-way algorithm to further balance the load of the two nodes.

The score function used here is the same as the score function used in the one way algorithm. It can also be generalized into the following form:

$$S(o, N_i) = \frac{\rho(o, N_1) + \rho(o, N_2)}{2} - \rho(o, N_i),$$

where $S(o, N_i)$ is called the score of operator o with respect to node N_i, $i = 1,2$. The intuition behind the use of $S(o, N_i)$ is that the larger the score, the better it is to put o on N_i instead of on the other node.

As this algorithm will move operators in both directions, it is called the *two-way correlation-based operator redistribution algorithm*.

The final mapping achieved by this algorithm can be much better than the one-way algorithm. However, as it does not consider the former locations of the operators, this algorithm tends to move more load than necessary, especially when the former mapping is relatively good. In the following section, we present an algorithm that can get a good operator mapping plan by only moving a small fraction of operators from the existing mapping plan.

3.3.3. Two-way Correlation-Based Selective Exchange.

In this algorithm, we allow both nodes to send load to each other. However, only the operators whose score is greater than certain threshold δ can be moved. The score function used is the same as the one in the one-way algorithm. Recall that if the score of an operator on node N_i , i = 1,2, is greater than zero, then it is better to put that operator on N_j $(j \neq i)$ instead of on N_i. Thus, by choosing $\delta > 0$, we only move operators that are good candidates. By varying the threshold δ, we can control the tradeoff between the amount of load moved and the quality of the resulting mapping plan. If δ is large, then only a small amount load will be moved. If δ is small (still greater than zero), then more load will be moved, but better mapping quality can be achieved.

The details of the algorithm are as follows: (1) Balance the load of the two nodes using the above one-way algorithm. (2) From the more loaded node[2], check whether there is an operator whose score is greater than δ. If so, move this operator to the less loaded node. (3) Repeat step (2) until no more operators can be moved or the number of iterations equals to the number of operators on the two nodes. (5) Balance the load of the nodes using the one-way algorithm.

As this algorithm only selects good operators to move, it is called *two-way correlation-based selective operator exchange algorithm*.

3.3.4 Improved Two-way Algorithms. In all above algorithms, operator migration is only triggered by load balancing. In other words, if an existing operator mapping plan is balanced, then no operator can be moved even if the load variance of some nodes is very large. To solve this problem and also maximize the average load correlation of the system, we add a correlation improvement step after each load balancing step in the above two-way algorithms.

Recall that if the load correlation coefficient of a node pair is small, then it is possible to further minimize the average load variance of the node pair. Thus, in the correlation improvement step, we move operators within a node-pair if their load correlation coefficient is below a certain threshold θ. Because we want to avoid unnecessary load migrations, the correlation improvement step is only triggered when some node is likely to get temporarily overloaded. The details of this step are as follows:

We define the "divergent load level" of each node as its average load plus its load standard deviation (i.e., square root of load variance). For each node with divergent load level more than one (it is likely to get temporarily overloaded), apply the following steps: (1) compute the load correlation coefficients between this node and all other nodes. (2) Select the minimum correlation coefficient. If it is less than θ, then apply one of the two way algorithms on the corresponding node pair (without moving the operators). (3) Compute the new correlation coefficient. If it is greater than the old one, then move the operators.

[2] The load of the nodes cannot be exactly the same.

Notice that this is only for the two-way algorithms since no operators can be moved in the one-way algorithm when load is balanced. The resulting algorithms are called *improved two-way algorithms.*

3.4. Global Operator Distribution

In this section we discuss a global algorithm which distributes all operators on n nodes without considering the former location of the operators. This algorithm is used to achieve a good initial operator distribution when the system starts. Because we need load statistics to make operator distribution decisions, the algorithm should be applied after a statistics collection warm up period.

The algorithm consists of two major steps. In the first step, we distribute all operators using a greedy algorithm which tries to minimize the average load variance as well as balance the load of the nodes. In the second step, we try to maximize the average load correlation of the system.

The greedy algorithm is similar to the one used in the two-way operator redistribution algorithm. This time, we start with n "empty" nodes (i.e., nodes with non-movable operators only). The movable operators are assigned to the nodes one by one. Each time, the node with the lowest load is selected as the receiver node and the operator with the largest score with respect to this node is assigned to it. Finally, the load of the nodes is further balanced using one round of the pair-wise one-way correlation-based load balancing algorithm.

The major difference between the global algorithm and the former pair-wise algorithm is that the score function used here is generalized to consider n nodes together. The score function of operator o with respect to Node N_i , i = 1, …, n, is defined as follows:

$$S(o, N_i) = \frac{1}{n}\sum_{j=1}^{n} \rho(o, N_j) - \rho(o, N_i),$$

The intuition behind $S(o, N_i)$ is that the larger the score, the better it is, on average, to put operator o on node N_i instead of putting it elsewhere. It is easy to verify that the score functions used in the pair-wise algorithms are just special cases of this form.

After all operators are distributed, a pair-wise correlation improvement step is then used to maximize the average load correlation of the system. First, we check whether the average load correlation of all node pairs is greater than a given threshold θ. If not, the node pair with the minimum load correlation is identified and the two-way operator redistribution algorithm is used to obtain a new mapping plan. The new mapping plan is accepted only if the resulting correlation coefficient is greater than the old one. Notice that if this process is repeated without change, the same node pair with the same set of operators on each node can be selected repeatedly. To avoid this problem, all selected node pairs are remembered in a list. When the process is repeated, only node pairs that are not in the list can be selected. If a new mapping plan is adopted by a node pair, then all node pairs in the list that include either of these nodes are removed from the list. This process is repeated until the average load correlation of the system becomes greater than θ or the number of iterations reaches the number of node pairs in the system.

4. Complexity Analysis

In this section, we analyze the computation complexity of the above algorithms and compare it with a traditional load balancing algorithm. The basic load balancing scheme of the two algorithms are the same. The later algorithm always selects operators with the largest average load first.

4.1. Statistics Collection Overhead

Assume each node has m operators on average and each load sample takes D bytes. Then the load statistics of each node takes $(m+1)kD$ bytes on average. Since the standard load balancing algorithm only uses the average load of each statistics window, the storage needed for statistics by the correlation based algorithm is k times that of the traditional load balancing algorithm.

On a high bandwidth network, the network delay for statistics transfer is usually negligible with regard to the load distribution time period. For example, we test the statistics transfer time on an Ethernet with 100Mbps connection between the machines. Establishing the TCP connection takes 2ms on average. When $m = 20$, $k = 20$, the statistics transfer time is 1ms per node on average. Considering the TCP connection time together with the data transfer time, the difference between the correlation based algorithm and the traditional load balancing algorithm is not significant.

4.2. Computation Complexity

First, consider the one-way correlation based load balancing algorithm. In each load distribution period, it takes $O(n\log n)$ time to order the nodes and select the node pairs. For a given node pair, before selecting each operator, the scores of the candidate operators must be computed. Computing the correlation coefficient of a time series takes time $O(k)$. Thus, in a pair-wise algorithm, computing the score of an operator also takes time $O(k)$. There are $O(m)$ operators on the sender node, thus the total operator selection time is at most $O(m^2k)$. In the traditional load balancing algorithm, it is not necessary to compute the scores of the operators, thus the operator selection time of the one-way correlation based algorithm is $O(k)$ times that of the traditional load balancing algorithm.

In the asymptotic sense, the two-way correlation based load balancing algorithms also takes time $O(m^2k)$ to redistribute the operators. But their computation time is several times that of the one-way algorithm as they consider twice as many operators as the later one considers.

For the global algorithm, the score computation takes $O(nk)$ time for each operator. As there are mn operators all together, its operator distribution time is $O(m^2n^3k)$. Thus the computation time of the greedy operator distribution step of the correlation based global algorithm is $O(nk)$ times that of the traditional load balancing algorithm.

Table 1: Computation time with different *n*

n	10	20	50
Computation Time	0.5sec	3.4sec	0.9min

Finally, consider the computation complexity of the correlation improvement steps. In the pair-wise algorithms, computing the divergent load level of all nodes takes time $O(nk)$. If a node is temporarily overloaded, selecting a node pair takes time $O(nk)$, and to redistribute load between them takes time $O(m^2k)$. There are at the most n temporarily overloaded nodes. Thus the whole process takes time at most $O(n^2k+ m^2nk)$

In the global algorithm, it takes time $O(n^2k)$ to compute the correlation matrix in the first iteration. In the following iterations, whenever operators are redistributed between a node pair, it take $O(nk)$ time to update the correlation matrix. Selecting a node pair takes time $O(n^2)$. Redistributing operators on a node pair takes time $O(m^2k)$. Thus, each complete iteration takes time $O(nk + n^2 + m^2k)$. There are at most $n(n-1)$ iterations. The total correlation improvement step takes time at most $O(n^3k + n^4 + m^2n^2k)$.

Although the correlation based algorithms are in polynomial time. They can still be very expensive when m, n, k are large. Thus, we must work with reasonable m, n, k to make these algorithms feasible.

4.3. Parameter Selection

Obviously, the global algorithm and the centralized pair-wise algorithm can not scale when n is large. However, we can partition the whole system into either overlapping or non-overlapping sub-domains. In each domain, both the global and the pair-wise algorithm can be applied locally.

In addition, as the pair-wise algorithm is repeated periodically, we must make sure that its computation time is small in comparison to the load distribution period. Obviously, when m is large, a lot of operators must have very small average load. As it is not necessary to consider each operator with small load individually, the operators can be clustered into super-operators such that the load of each super-operator is no less than certain threshold. By grouping operators, we can control the number m on each node.

Moreover, we can also choose k to achieve a tradeoff between the computation time and the performance of the algorithm. For larger k, the correlation coefficients are more accurate, and thus the distribution plans are better. At the other extreme, when k is 1, our algorithm reduces to load balancing with a random operator selection policy.

Finally, we would like to point out that it is not hard to find reasonable m, k, and domain size n. For example, we tested the algorithms on a machine with an AMD Athlon™ 3200+ 2GHz processor and 1GB memory. When m=10, k=10, the computation time of the pair-wise operator redistribution algorithm is only 6ms for each node pair. If the load distribution interval is 1 second, the pair-wise algorithms only take a small fraction of the CPU time in each distribution period. Since the pair-wise algorithm can be easily extended to a decentralized and asynchronous implementation, it is potentially scalable. The computation time of the global algorithm with different *n* is shown in Table 1. Note that the global algorithm runs infrequently and on a separate node. It would only be used to correct global imbalances.

5. Experiments

In this section, we present experimental results based on a simulator that we built using the CSIM library [12].

5.1. Experimental Setup

5.1.1. Queries. For these experiments, we use several independent linear operator chains as our query graphs. The selectivity of each operator is randomly assigned based on a uniform distribution and, once set, never changes. The execution cost of each operator is at most 0.1 second. We also treat all operators in the system as movable.

5.1.2. Workload. We used two kinds of workloads for our experiments. The first models a periodically fluctuating load for which the average input rate of each input stream alternates periodically between a high rate and a low rate. Within each period, the duration of the high rate interval equals the duration of the low rate interval. In each interval, the inter-arrival times follow an exponential distribution with a mean set to the current average data rate. We artificially varied the load correlation coefficients between the operators from –1 to 1 by aligning data rate modes of each input stream with a different offset.

The second workload is based on the classical On-Off model that has been widely used to model network traffic [3, 18]. We further simplified this model as follows: each input stream alternates between an active period and an idle period. During an active period, data arrives periodically whereas no data arrives during an idle period. The durations of the active and idle periods are generated from an exponential distribution. This workload models an unpredictably bursty workload. In order to get different load correlations from -1 to 1, we first generate some input streams independently and then let the other input streams be either the opposite of one of these streams (when steam *A* is active, its opposite stream is idle and vise versa) or the same as one of these streams with the initial active period starting at a different time.

We use the periodically fluctuating workload to evaluate the global algorithm alone and to compare the pair-wise algorithms with the global algorithm. The bursty workload is used to test both algorithms together, as the global load distribution easily becomes ineffective under such workload.

5.1.3. Algorithms. We compare the above correlation based algorithms with a traditional load balancing algorithm which always selects the operator with largest load first, and a randomized load balancing algorithm which randomly picks the operators. Each of the latter two algorithms has both a global version and a pair-wise version. Operators are only moved from the more loaded nodes to the less loaded nodes.

Table 2: Simulation Parameters

Number of nodes (*n*)	20
Average # of operators per node (*m*)	10
Number of operators in each chain	10
Operator selectivity distribution	U (0.8, 1.2)
Operator processing delay (per tuple)	1ms
Input rate generating distribution	U(0.8, 1.2)
Input rate fluctuation period	10sec
Input rate fluctuation ratio (high rate/low rate)	4
Operator migration time	200ms
Network bandwidth	100Mbps
Statistics window Size	10sec
# of samples in statistics window (*k*)	10
Load distribution period	1sec
Load balancing threshold (ε)	0.1
Score threshold for operator exchange (δ)	0.2
Correlation improvement threshold (θ)	0.8

5.1.4 Experiments. Unless specified, the operators are randomly placed on all nodes when a simulation starts. All experiments have an initial warm up period, when the load statistics can be collected. In this period, a node only offloads to another node if it is overloaded. The receiver node is selected using the same algorithm described in Section 3.3. After the warm up period, different load distribution algorithms are applied and the end-to-end latencies at the output are recorded.

We test each algorithm at different system load levels. The *system load level* is defined as the ratio of the sum of the busy time of all nodes over the product of the number of nodes and the simulation duration. For each simulation, we first determine the system load level, then compute the average rate of each input streams (to achieve the given load level) as follows: (1) Randomly generate a rate from a uniform distribution. (2) Compute the system load level using the generated steam rates. (3) Multiply each stream rate by the ratio of the given system load level over the computed system load level.

To avoid bias in the results, we repeated each experiment five times with different random seeds, and we report the average result. In order to make the average end-to-end latency of different runs comparable, we make each operator chain contain the same number of operators each with the same processing delay. In this setting, the end-to-end processing delay of all output tuples is the same. (i.e., no dependency on the randomly generated query graph).

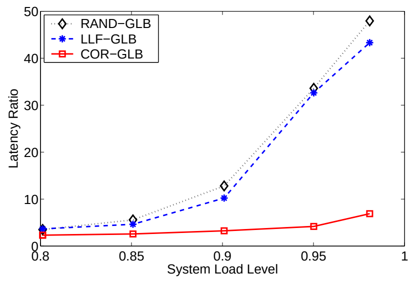

Figure 3: Latency ratio of the global algorithms

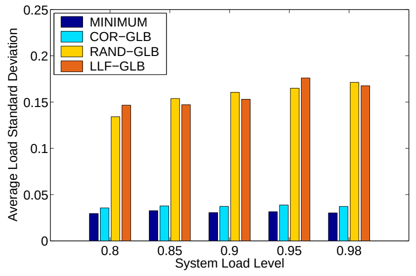

Figure 4: Average load variance of the global algorithms

Because the average end-to-end latency depends on the number of operators in each chain as well as the processing delay of each operator, we use the ratio of the average end-to-end latency over the end-to-end processing delay as the normalized performance measurement. This ratio is called the *latency ratio*.

Unless otherwise specified, all the experiments are based on the simulation parameters summarized in Table 2.

5.2. Experiments and Results

5.2.1. The Global Algorithms. First, we compare the three global operator allocation algorithms. They are the *correlation based algorithm* (COR-GLB), the *randomized load balancing algorithm* (RAND-GLB) and the *largest-load-first load balancing algorithm* (LLF-GLB).

In the first experiment, the global algorithms are applied after the warm up period and no operator is moved after that. The latency ratios of these algorithms at different system load levels are shown in Figure 3. Obviously, the correlation based algorithm performances much better than the other two algorithms. Figure 4 depicts the average load standard deviation of all nodes in the system after the global algorithms are applied. The COR-GLB algorithm results in load variance that is much smaller than the other two algorithms. This further confirms that small load variance leads to small end-to-end latency. We also show the lower bound of the average load standard deviation (marked by MINIMUM) in Figure 4. It is the standard deviation of the overall system load time series divided by *n* (according to Theorem 1). The results show that the average load variance of the COR-GLB algorithm is very close to optimal in this experiment.

In addition, we measured the average load correlation of all node pairs after the global distributions. The results of

Table 3: Average load correlation of the global algorithms

COR-GLB	RAND-GLB	LLF-GLB
0.65	-0.0048	-0.0008

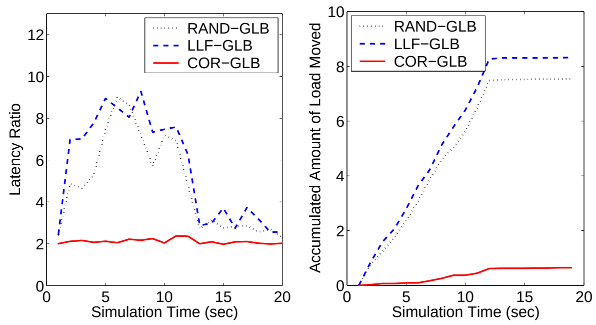

Figure 5: Dynamic performance of the global algorithms

one algorithm at different load levels are similar to each other and the average results are shown in Table 3. Notice that the average load correlation of the RAND-GLB and the LLF-GLB algorithms are around zero, showing that their performance is not worst case. If an algorithm tends to put highly correlated operators (for instance, connected operators with fixed selectivity) together, it may result in an average load correlation close to -1. This would get much worse performance under a fluctuating workload.

The benefit of having large average load correlation is not obvious in the first experiment. The above results seem to indicate that when the system load level is lower than 0.5, it does not matter which algorithm is used. However, this is not true. In the second experiment we show the effect of the different average load correlations achieved by these algorithms.

In this experiment, we first set the system load level to be 0.5 and use different global algorithms to get initial operator distribution plans. Then, we increase the system load level to 0.8 and use the largest-load-first pair-wise load balancing algorithm to balance the load of the system. The latency ratios and the *amount of load moved* [3] after the load increase are shown in Figure 5. Because the COR-GLB algorithm results in large average load correlation, the load of the nodes is naturally balanced even when the system load level changes. On the other hand, the RAND-GLB and the LLF-GLB algorithms are not robust to load changes as they only have average load correlations around zero. Therefore, the correlation based algorithm is still potentially better than the other two algorithms even if the current system load level is not high.

5.2.2. The Pair-wise Algorithms. For the pair-wise algorithms, we want to test how fast and how well they can adapt to load changes. Thus, in the following experiments, we let the system start from connected mapping plans where a connected query graph is placed on a single node.

[3] Whenever an operator is moved, its average load is added to the amount of load moved.

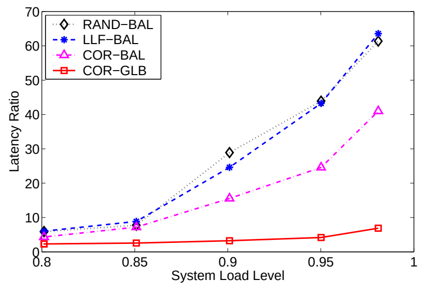

Figure 6: Latency ratio of the one-way pair-wise algorithms and the correlation based global algorithm

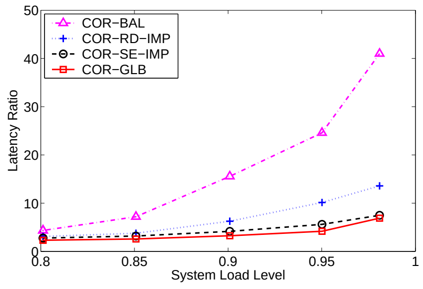

Figure 7: Latency ratio of correlation based algorithms

Different pair-wise algorithms are applied after the warm up period and the worse case recovery performance of these algorithms is compared.

One-way Pair-wise Load Balancing Algorithms: First the three one-way pair-wise algorithms are compared. They are the *correlation based load balancing algorithm* (COR-BAL), the *randomized load balancing algorithm* (RAND-BAL) and the *largest-load-first load balancing algorithm* (LLF-BAL). Figure 6 depicts the latency ratios of these algorithms at different system load levels. Obviously, the COR-BAL algorithm has the best performance. Because the amount of load moved for these algorithms is almost the same, the result indicates that the operators selected by the correlation base algorithm are better than those selected by the other two algorithms. The latency ratios of the correlation based global algorithm are added in Figure 6 for comparison. It shows that the performance of these pair-wise algorithms is much worse than that of the correlation based global algorithm.

Improved two-way pair-wise algorithms: In this experiment, we compare two improved correlation based two-way algorithms. They are the *improved operator redistribution algorithm* (COR-RE-IMP) and the *improved selective operator exchange algorithm* (COR-SE-IMP). The latency ratios of the COR-BAL and the COR-GLB algorithms are added in Figure 7 for comparison. The results show that the latency ratios of the improved two-way pair-wise algorithms are much smaller than the one-way algorithm. Thus, the benefit of getting better operator distribution plans exceeds the penalty of moving more operators.

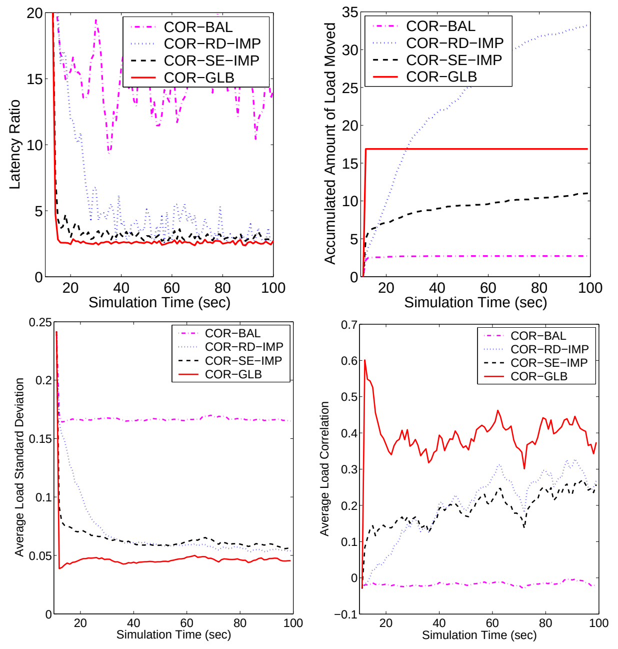

Figure 8: Performance of the correlation based algorithms when system load is 0.9

To look at these algorithms more closely, we plot several metrics with respect to the simulation time when the system load level is 0.9 in Figure 8. Obviously, the COR-RE-IMP algorithm moves much more load than the COR-SE-IMP algorithm. Thus although the quality of its final plan is closer to that of the global algorithm, its average performance is worse than that of the COR-SE-IMP algorithm. For different applications, which two-way algorithm performs better on average usually depends on the workload of the system and the operator migration time.

We can also see from Figure 8 that the global algorithm moves less load than the COR-RE-IMP algorithm but achieves better performance. Thus, although it is possible to use pair-wise algorithms only, it is still sensible to use a global algorithm for initial operator distribution.

5.2.3. Sensitivity Analysis. Here, we inspect whether the correlation based algorithms are sensitive to different simulation parameters. In these experiments, the COR-GLB and the COR-SE-IMP algorithms are compared with the LLF-GLB and the LLF-BAL algorithms when the system load level is 0.9. We vary the number of nodes (*n*), the average number of operators on each node (*m*), the size of the statistics window, the number of samples in each statistics window (*k*), the input rate fluctuation period, and the input rate fluctuation ratio (high rate / low rate).

The results in Figure 9 show that the correlation based algorithms are not sensitive to these parameters except when *m* is very small, in which case, the load of the system cannot be well balanced. On the other hand, the largest-load-first load balancing algorithms are sensitive to these parameters. They perform badly especially when the number of nodes is small, or the average number of

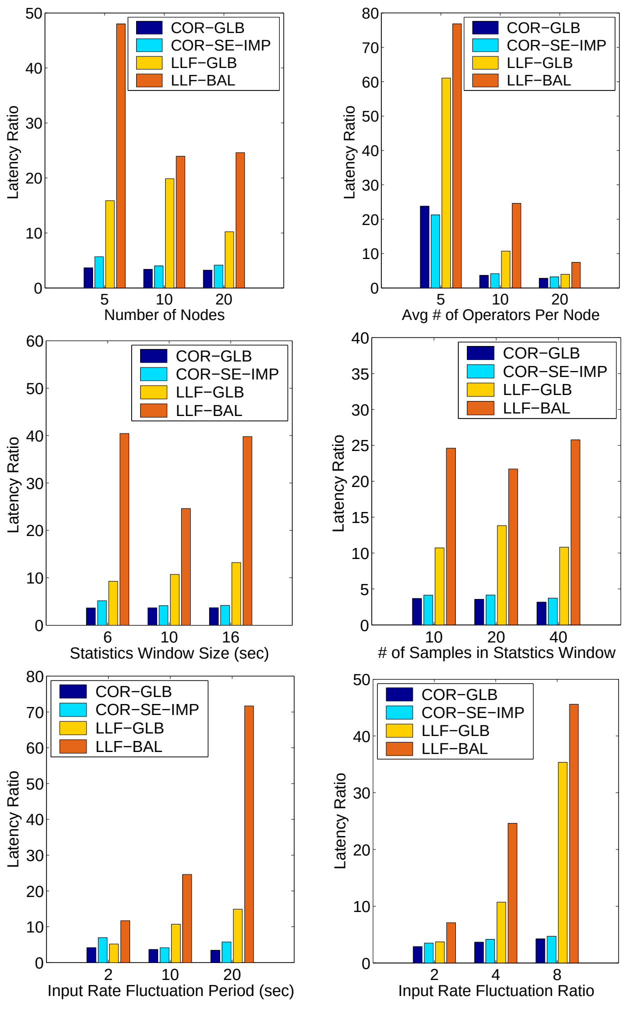

Figure 9: Experiments with different parameters

operators on each node is small, or the load fluctuation period is long, or the load fluctuation ratio is large,

Notice that when *m* is large, the static performance of the largest-load-first algorithm is almost as good as the correlation based algorithms. This is because when each operator has only a small amount of load and the load of all operators fluctuate differently, putting a lot of operators together can smooth out load variation. However, when the dynamic performance is considered, the correlation based algorithm still performs better than the largest-load-first algorithm because it results in a positive average load correlation and can naturally balance the load when the load changes.

In addition, these results show that the correlation based algorithms are not very sensitive to the precision of the measured correlations. They work pretty well even when the size of the statistics window is only half of the load fluctuation period (i.e., when load fluctuation period is 20 in Figure 9). Thus, when the precision of the load correlations must be sacrificed to reduce the computation overhead, we can still expect relatively good performance.

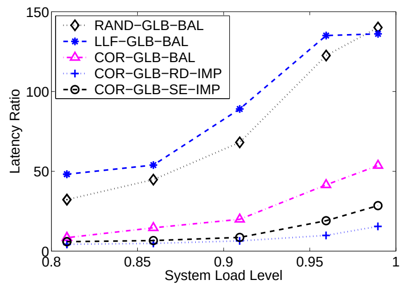

Figure 10: Latency ratio of different algorithms with on-off input model

5.2.4. Bursty Workload. Finally, we use the bursty workload to test the robustness of our algorithms. The mean of the active period durations and the mean of the idle periods are both 5 seconds, and the statistics window size is still 10 seconds. As the duration of the active periods and the idle periods are exponentially distributed, the measured load correlations vary over time, and they are not precise.. In this experiment, the global algorithms are combined with their corresponding pair-wise algorithms. The combined algorithms are identified by the names of the pair-wise algorithms with GLB inserted. The experimental results in Figure 10 confirm the effectiveness of the correlation based algorithms under such workload.

6. Related Work

Load distribution is a classical problem in distributed and parallel computing systems [7, 11, 20]. In most of the traditional systems, load balancing or load sharing is achieved by wisely allocating new tasks to processing units before their execution [16]. Due to the high overhead of load migration, the applications of dynamic load distribution algorithms (which redistribute running tasks on the fly) are usually restricted to large scientific simulations and computations [15, 17]. Stream based data processing systems [2, 5, 13] are different from traditional database systems in that they are push-based and the tasks in these systems are continuous queries. Because the input data rates of such systems do not depend on the resource utilization, the load distribution algorithms for these systems are also different from traditional works.

Dynamic load balancing has been studied in the context of continuous query processing. Shah et al. studies how to process a single continuous query operator on multiple shared-nothing machines [14]. In this work, load balancing is achieved by adjusting the data partitions on the servers dynamically. Our work is complementary to theirs since we focus on inter-operator load distribution instead of intra-operator data partition.

Our previous work [19] studies dynamic load distribution in stream processing systems when the network transfer delays are not negligible. In this work, connected operators are clustered as much as possible to avoid unnecessary network transfers. When load redistribution is necessary, operators along the boundary of the sub-query graphs are migrated in order to achieve a good balance between the operator distribution quality and the load migration overhead. Our current work is based on different assumptions where we consider frequently fluctuating workloads with abundant network resources.

Another dynamic load management algorithm for distributed federated stream processing systems is presented in [4]. In this system, the autonomous participants do not collaborate for the benefit of the whole system. A price must be paid if one node wants to offload to another node. Using pre-negotiated pair-wise contracts, these participants can handle each other's excess load. Our work is different from this work in that we consider stream processing servers in the same administrative domain where all nodes fully cooperate with each other. In addition, our algorithm considers the load variation of the operators and tries to find load distribution plans with small average load variance and large average load correlation. To the best of our knowledge, this problem has not been addressed by any of the former work yet.

7. Conclusions and Future Directions

We have studied in-depth a class of algorithms that statically finds a good initial operator placement in a distributed environment and that dynamically moves operators to adapt to changing loads. We have shown that by considering load correlations and load variations, we can do much better than conventional load balancing techniques. This illustrates how the streaming environment is fundamentally different from other parallel processing approaches. The nature of the operators and the way that data flows through the network can be exploited, as we have, to provide a much better solution for minimizing end-to-end latency.

The work presented here focuses on high-performance computing clusters such as blade computers. An obvious direction for future work is to relax this constraint, and to move toward a more heterogeneous computing environment in which bandwidth and power consumption are important resources that must be conserved as well. This will radically change the optimization algorithms. We believe that by starting with the more familiar and, in its own right, useful case in this study, we will be better informed to tackle the next set of problems.

8. References

[1] D. Abadi, Y. Ahmad, H. Balakrishnan, M. Balazinska, U. Cetintemel, M. Cherniack, J. Hwang, J. Jannotti, W. Lindner, S. Madden, A. Rasin, M. Stonebraker, N. Tatbul, Y. Xing, S. Zdonik, The Design of the Borealis Stream Processing Engine. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR),* Jan. 2005.

[2] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul and Stan Zdonik. Aurora: A New Model and Architecture for Data Stream Management. *VLDB Journal,* Sep. 2003.

[3] A. Adas, Traffic models in broadband networks. *IEEE Communications*, 35(7):82--89, July 1997.

[4] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-based load management in federated distributed systems. In *USENIX Symposium on Net-worked Systems Design and Implementation (NSDI),* March 2004.

[5] S. Chandrasekaran, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. of the CIDR Conference,* Jan. 2003.

[6] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing and S. Zdonik. Scalable Distributed Stream Processing. In *Proc. of the CIDR Conference,* 2003.

[7] R. Diekmann, B. Monien, and R. Preis, Load balancing strategies for distributed memory machines. *Multi-Scale Phenomena and Their Simulation*, 255--266. World Scientific, 1997

[8] A. Foong, T. Huff, H. Hum, J. Patwardhan, G. Regnier, TCP performance re-visited. In *Proc. of IEEE Intl Symposium on Performance of Systems and Software,* March 2003.

[9] A. Gallatin, J. Chase, and K. Yocum, Trapeze/IP: TCP/IP at near-gigabit speeds. In *Proc. of USENIX Technical Conference*, June 1999.

[10] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.

[11] D. Gupta and P. Bepari, Load sharing in distributed systems, In *Proc. of the National Workshop on Distributed Computing*, January 1999.

[12] Mesquite Software, Inc. CSIM 18 Simulation Engine. http://www.mesquite.com/

[13] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system. In *Proc. of the CIDR Conference,* 2003.

[14] M.A. Shah, J.M. Hellerstein, S. Chandrasekaran, and M.J. Franklin. Flux: An Adaptive Partitioning Operator for Continuous Query Systems. In *Proc. of the ICDE Conference*, pages 25--36, 2003.

[15] K. Schloegel, George Karypis and Vipin Kumar. Graph Partitioning for High Performance Scientific Simulations. *CRPC Parallel Computing Handbook*. Morgan Kaufmann, 2000.

[16] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. Scheduling and load balancing in parallel and distributed systems. IEEE Computer Science Press, 1995.

[17] C. Walshaw, M. Cross, and M. G. Everett, Dynamic load balancing for parallel adaptive unstructured meshes. *Parallel Processing for Scientific Computing,* 1997. 10

[18] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson, Self-similarity through high variability: statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71--86, 1997.

[19] Ying Xing. Load Distribution for Distributed Stream Processing. In *Proc. of the ICDE Ph.D. Workshop,* 2004.

[20] C. Xu, B. Monien, R. Luling, and F. Lau. Nearest neighbor algorithms for load balancing in parallel computers. *Concurrency: Practice and Experience*, 9(12):1351--1376, 1997.