

# Dynamic Load Management for Distributed Continuous Query Systems

Yongluan Zhou      Beng Chin Ooi      Kian-Lee Tan  
National University of Singapore  
{zhoyong, ooibc, tankl}@comp.nus.edu.sg

## 1. Introduction

In many emerging monitoring applications (e.g. network management, sensor networks, financial monitoring etc.), data occurs naturally in the form of active continuous data streams. These applications typically require the processing of large volumes of data in a responsive manner. Moreover, the sources of data streams are naturally distributed (such as sensors in a sensor network). In order to scale up the volumes of streams and queries that can be processed, a distributed stream processing system is inevitable. However, as the properties of data streams (e.g., arrival rates) and the processing servers' load are hard to predict, the initial placement of query operators may result in unsatisfactory system performance. The problem is exacerbated by multiple continuous queries that run long enough to experience the changes of the environment parameters. As such, any suboptimal performance will persist for a long time.

Clearly, a distributed stream processing system must adapt to changes in environment parameters and servers' load. We believe a dynamic load management scheme is indispensable for the system to be scalable. In particular, we expect aggressive methods such as query operator migration during runtime to bring long term benefit (especially for long running continuous queries) even though they may incur some short term overhead. However, to date few complete and practical solutions have been proposed for this problem. In this paper, we offer our solution to the problem. More specifically we make the following contributions:

- We formally define a new metric, *Performance Ratio (PR)*, to measure the relative performance of each query and the objective for the whole system.
- By building a new cost model, we identify the heuristics that can be used to approach the objective.
- We propose a complete and practical distributed load management scheme which includes a static initial placement scheme for newly initiated queries as well as a runtime dynamic scheme.
- We conducted an extensive experimental study that shows the effectiveness of our technique.

## 2. Problem Formulation and Analysis

In our system, there is a set of stream sources, and a set of locally distributed processing nodes and a set of continuous queries. Since stream sources may not have the ability to communicate with multiple nodes, we assign one processing node as the delegation of each stream source. In terms of system performance, we are concerned about the delay of resulting data items, which is also one of the main concerns of end users. Formally, if the evaluation of query  $q_k$  on a source tuple  $tuple_l$  from  $s_l$  generates one or more result tuples, then the delay of  $tuple_l$  for  $q_k$  is defined as  $d_k^l = t_{out} - t_{in}$ , where  $t_{in}$  is the time that  $tuple_l$  arrived at the system and  $t_{out}$  is the time that the result tuple is generated. If there are more than one result tuples, then  $t_{out}$  is the time that the last one is generated. At a closer look,  $d_k^l$  includes the time used in evaluating the query (denoted as  $p_k^l$ ), the time waiting for processing as well as the time it is transferred over the network connections. For a specific processing model, we assume the evaluation time  $p_k^l$  is fixed and regard it as the inherent complexity of  $q_k$ . Since different queries may have different inherent complexities, the value of  $d_k^l$  cannot reflect correctly the relative performance of different queries. However, in a multi-query and multi-user environment, we wish to tell the relative performance of different queries. Hence we propose a new metric *Performance Ratio (PR)* to incorporate the inherent complexity of a query. Formally,  $PR_k^l$  for processing of  $tuple_l$  for  $q_k$  is defined as

$$PR_k^l = \frac{d_k^l}{p_k^l}. \quad (1)$$

And the performance ratio of  $q_k$  is defined as

$$PR_k = \max_{s_l \in S_k} PR_k^l. \quad (2)$$

$PR_k$  reflects the relative performance of  $q_k$ . Our objective is to minimize the worst relative performance among all the queries.

In [1], we developed a cost model to estimate the values of  $d_k^l$  and  $p_k^l$ . And hence the value of  $PR_k$  can be calculated accordingly. Given the cost model, we derived

that the problem is actually NP-Complete. Furthermore, we also observed that if we ignore the communication cost, the workload should be balanced across all the processing nodes in an optimal solution. In view of the complexity of the problem, we opt to designing heuristics instead of finding an optimal algorithm. From the cost model, we know that the extra delay is caused by the communication and the workload of the system. Hence, we use the heuristics as follows. (1) Distribute operators of a query to a restricted number of nodes so that  $CPR_k^l$  is small. As we will see soon, we set the maximum of this number as the number of streams that the query operates on. The intuition is the processing cost is higher for a query that involves more streams, and hence it can afford to be distributed to more nodes. (2) Dynamically balance the workload of the processing nodes. This heuristic is inspired by our observation stated above. (3) Minimize the communication cost under conditions (1) and (2). In short, we have to design a dynamic load balancing scheme where the operations of each query should not be distributed to too many nodes and the total communication traffic is minimized.

### 3. System Design

**Initial placement scheme.** In our initial placement scheme, we only consider minimizing the communication cost and leave the load balancing task to our dynamic scheme. The scheme generates one query fragment for each involved stream and then distributes them to the delegation nodes of their corresponding streams. Roughly speaking, the scheme places the filters of a stream at that stream's delegation node and places the join operators at nodes that minimize communication traffic. Detail of the query fragment generation algorithm can be found in [1]. We further defined two query fragments are neighbors to each other if there is data flow between them.

**Dynamic placement scheme.** In the dynamic placement scheme, we adopt a local load balancing approach. Each node  $n_i$  has a number of load management partners. The partner relationship is symmetric, i.e. if  $n_i$  is a partner of  $n_j$ , then  $n_j$  is also a partner of  $n_i$ . Each node would keep track of the load distribution within itself and all its partners and would make load redistribution decisions if necessary. In the partner selection algorithm described in [1], one node tends to select those nodes that contain more neighboring query fragments of those query fragments in this node.

During runtime, each node would collect its own workload within a window time and if it detects the workload changes to some degree, it will broadcast to all its partners. This basic technique suffers from transient changes of the system state, which would render the system unstable. To solve this problem, we adopt a low pass filter to filter out the transient changes of the system workload. In particular, workload is computed as  $\rho_{i+1} = \alpha \times \rho_i + (1 - \alpha) * \rho_c$ ,

where  $\rho_{i+1}$  and  $\rho_i$  are the workload information used for load balancing after  $i + 1$  and  $i$  time windows, and  $\rho_c$  is the collected workload within the  $(i + 1)$ th time window.  $\alpha$  is a parameter to determine the responsiveness of the estimation value to the workload changes. If  $\alpha$  is too high then the calculated workload cannot reflect the current workload. On the other hand, if  $\alpha$  is too low then it cannot filter out the transient changes. In [1], we mathematically analyzed how to set  $\alpha$  in practice.

Based on the load information of itself and its partners, each node would make load management decisions at runtime. This works in rounds. Within each round, if a node detects that its own workload is smaller than the average workload of itself and its partners, it would request workload from the partner that has the largest workload.

Once a node receives the workload request, it will choose the victim query operators for migration. In the design of this selection scheme, we have to consider two points. Firstly, in the perspective of system performance, the selection scheme should work fast and scalable to the number of queries. Secondly, we have to restrict the number of processing nodes for a query to the number of involved streams. Hence, we use the query fragments generated in the initial placement scheme as our smallest migration unit. The number of query fragments is equal to the number of involved streams. Hence using query fragment as migration unit inherently restricts the number of processing nodes for a query. Moreover, we expect this number would be much smaller than the number of operators and hence the load selection decision can be made much faster. Furthermore, the choice of query fragments to be migrated is critical in maintaining data flow locality. A poor choice may cause streams to be scattered across too many nodes and result in network congestion. In [1], we proposed a data flow aware strategy, which performs well in maintaining data flow locality.

We presented extensive experimental results in [1]. In the experiments, we found that the proposed strategy works well with a small number of load management partners, which can minimize the runtime overhead. We also examined our data flow aware load selection scheme by comparing it with two simpler algorithms. The results showed that the former works better in maintaining data flow locality than the others. Finally we studied the adaptivity, stability and sensitivity of our scheme to changes in stream arrival rates, system workload and the parameter  $\alpha$ . The results showed that our scheme can effectively adapt to the runtime changes of the system to approach our objective.

### References

- [1] Y. Zhou, B.C. Ooi, and K.-L. Tan. Dynamic Load Management for Distributed Continuous Query Systems. *Unpublished manuscript available from authors*, 2004.