Operator Placement Techniques for Distributed Stream-Processing Systems

Min Chen, Danhua Guo
{michen, dguo}@cs.ucr.edu
CS Department, UC Riverside

Dec 8th, 2006

Abstract

The idea of pushing query operators to nodes within the network helps distributed stream-processing systems to use their pool of resources more efficiently than ever. However, determining placement locations is challenging because of the fact that network conditions changes over time and streams may interact with each other. This project describes a Stream-Based Overlay Network (SBON) with embedded operator placement algorithms to enable the reusing and repositioning of operators, and therefore improves network utilization and lowers stream latency. We will compare the result of relaxation placement in SBON with other related operator placement algorithms in terms of placement efficiency and optimization. The result and evaluation of our operator placement algorithm is provided by the end of the paper.

1 Introduction

In this project, we implemented and evaluated a *Network-Aware Operator Placement algorithm*[1] for stream-processing systems. The algorithm places operators using a spring relaxation technique that manages single- and multi-query optimization. The implementation will be based on a *stream-based overlay network (SBON)*, a layer between a stream-processing system and the physical network that manages operator placement for stream-processing systems[1]. We presented the evaluation of the operator placement algorithm through software simulation. By comparing the result with other placement schemes, we investigated and evaluated the placement efficiency in network usage and delay penalty.

2 Background and Related Work

Stream-based continuous query processing fits a large class of new applications, such as sensor networks, location-tracking services, network management, fabrication line management and financial data analysis. Data generated from external sources is pushed asynchronously to servers that process the information in these systems, which are characterized by the need to process high-volume data streams in a timely and responsive fashion. As data flows from its point of origin downstream to applications, it passes through many computing devices, each of which is a potential target of computation. Furthermore, many servers would be brought to cope with time-varying load spikes and balancing demand. Therefore these applications could benefit substantially from the power of distributed systems. We refer to such applications as stream-based applications and systems as distributed stream-processing system(DSPS)[2].

A distributed stream-processing system(DSPS) streams data from multiple producers to multiple consumers via in-network processing operators, eliminating centralized processing[1]. The DSPS instantiates the query in the network in the form of a query stream of data tuples flowing through the network. Operators that are part of a query stream are interconnected by overlay operator links, each with a certain latency and a data rate. One of the main tasks of DSPSs is operator placement, or the selection of the physical node that should host the operator.

Basically DSPSs have to face several challenges when solving the operator placement problem[1]. First, applications have multiple QoS demands including high throughput, small delay and jitter. To obtain good query performance for applications becomes very important. Second, applications are composed at run-time, without a priori notification, posing stringent resource requirements on processor cycles and available network bandwidth along the streaming paths. Designers have to find a way to use the network resources efficiently, and if possible find and reuse the existing operators under some conditions. As a result, resource allocation becomes an important factor that affects the scalability of service overlays and the performance of the applications when deployed on a shared and heterogeneous infrastructure. Although DSPSs have the need for decentralized, efficient operator placement and operator reuse, no common infrastructure exists for optimizing distributed data streams in a network-aware fashion.

Allocation of processes, tasks or services has been a long studies topic in a variety of distributed systems, including distributed operating systems, and databases. Recent efforts have studied this topic in distributed stream processing environments [3], [2], [4]. In [2] the slide and split operations are introduced for load sharing. Not only do these facilitate similar but fine-grained load balancing, but also take into account operator dependencies, properly splitting and merging the input and resulting data streams as necessary. In [4] load variance is minimized by maximizing load correlation. Their approach consists of a centralized algorithm for initial load distribution and a pair-wise algorithm for dynamic load migration. [3] proposes an adaptive and scalable load balancing technique for fair allocation of resources in large-scale service overlays, so that QoS demands of distributed stream processing application are satisfied. This approach allows composition of distributed stream processing applications dynamically, to satisfy their end-to-end QoS demands with high probability. And it distributes the load across multiple peers, driven by decisions made by the individual nodes and based on information they have stored locally. From these work we could see that integrated monitoring, dynamic execution time prediction and scheduling across multiple processors become very important for adaptive systems. And efficient dynamic balancing algorithms require knowledge of real time application task information including the application task's deadline, reliability and resource requirements.

Most DSPSs, such as Borealis[4], currently avoid the operator placement problem by supporting only pre-defined operator locations with pinned operators in the network. This leaves the burden of efficient operator placement to the system administrator, which is infeasible for a dynamic, large-scale system with thousands of queries. Other DSPSs, such as Medusa[2], place operators to improve application performance by balancing load. This is appropriate within a single data center but leads to poor performance on a wide-area network, in which communication latencies can dominate processing costs. In addition, these approaches are not scalable to DSPSs with thousands of nodes that must be considered for placement. [1] solve the operator placement problem with a two step approach: (1) the placement decision is made in a virtual cost space, and (2) the cost space coordinate is mapped to a physical node. The cost space encodes network and node measurements from all nodes in a scalable and decentralized manner, enabling individual nodes to make adaptive placement decisions with local information.

3 Network-Aware Operator Placement

3.1 Network Usage

Because of the three-fold challenges faced up with operator placement problems, the way we used to measure the efficiency of an operator placement algorithm should try to consider all of these aspects.

From an individual application perspective, users do not want their query to take forever to execute. Some users even pay relatively more to be guaranteed to receive a satisfactory response as soon as possible. Therefore the latency of each query become the most important factor for such kind of services. On the other hand, judging from the network as a whole, we need to make sure that the network has been fully utilized, namely trying to reaching the minimization of "idled" network resources. However, the less network resources distributed, the less confidence we have to handle the latency issue. [1] introduces a Network Usage metric to balance between individual query latency with consumed network bandwidth. Our simulation results prove the efficiency of this metric.

Basically the idea of Network Usage is to trade off the overall application delay and network bandwidth consumption. The network usage $u(q)$ is the amount of data that is in-transit for query q at a given instant:

$$u(q) = \sum_{l \in L} DR(l) \cdot Lat(l)$$

where L is the set of links used by the stream, DR(l) is the data rate over link l, and Lat(l) is the latency. This captures the bandwidth-delay product of the query. This captures the bandwidth-delay product of the query. The network usage U for the entire DSPS is simply the sum over all queries in the system.

While high latency links are obviously poor choices for latency-sensitive applications, they are poor choices for networking reasons as well. High latency between two nodes indicates that the routing path traverses a long geographical distance or passes through multiple physical links with intermediate routers. In both cases, the monetary cost of operating such a path is likely higher compared to shorter ones. Also, if path congestion is causing a links high latency, this link should be avoided for the global good.

3.2 Cost Space

A cost space is a d-dimensional metric space where the Euclidean distance between two nodes is an estimate of the cost of routing data between those nodes[1]. Each SBON node is responsible for maintaining its own coordinate. By projecting the placement problem into a virtual cost space and then mapping the solution back to physical nodes, SBON nodes can compute operator placement decisions in a continuous mathematical space with the use of sophisticated optimization techniques.

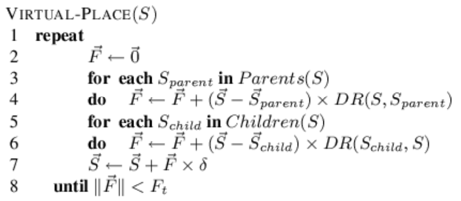
Many performance metrics can be used to define a cost space. Generally we examine latency and load. A latency space can be constructed using a network embedding[5] that embeds measured latencies between nodes in a low-dimensional geometric space, with the goal that the Euclidean distance between two network coordinates is an estimate of physical network latency. The advantages of such space are 1) the cost to maintain a latency space is small because a network coordinate could be obtained after the node probes the latency to only a small subset of nodes. 2) Simulation results show that the network coordinates have a small prediction error with a media of 11%, even though Internet latencies often violate the triangle inequality and change dynamically over time.

Two classes of algorithms were proposed to compute latency network coordinates[5]: landmark-based schemes and simulation-based approaches. The Vivaldi algorithm of the latter is used in SBON, which is a fully-decentralized, scalable method compared to others.

3.3 Virtual Operator Placement

The operator placement technique, called RELAXATION, is the core algorithm in the SBON system[1]. It makes efficient operator placement decisions within the cost space. The main idea behind RELAXATION is to divide the placement problem into two steps. First, an unpinned operator in a query is placed using a spring relaxation technique in the 3-dimensional latency cost space, then the solution is mapped to the closest physical node in the cost space. The port-mapping from virtual to physical node will be explained in the next section.

We could consider queries a collection of massless bodies (operators) connected by springs (operator links). Every pinned operator has a fixed location, while unpinned operators can move to anywhere. Thus operator placement problem then becomes minimizing the energy state of the system, namely minimizing the sum of the potential energies stored in the springs. After some mapping[1], the operator placement metric, network usage could be included in the cost function, and there is a unique solution from a set of placement even with equal network usage. 1 shows us the pseudocode of RELAXATION.



```
VIRTUAL-PLACE($S$)
repeat
        $\vec{F} \leftarrow \vec{0}$
        for each $S_{parent}$ in $Parents(S)$
        do  $\vec{F} \leftarrow \vec{F} + (\vec{S} - \vec{S}_{parent}) \times DR(S, S_{parent})$
        for each $S_{child}$ in $Children(S)$
        do  $\vec{F} \leftarrow \vec{F} + (\vec{S} - \vec{S}_{child}) \times DR(S_{child}, S)$
        $\vec{S} \leftarrow \vec{S} + \vec{F} \times \delta$
    until $\|\vec{F}\| < F_t$
```

Figure 1: RELAXATION Algorithm

3.4 Physical Operator Placement

After the optimal solution in the virtual cost space is found, we need to map the target cost space coordinate to a physical node. We do not need to consider the resource status, like load, bandwidth, etc. in virtual operator placement because we hope to find the optimal solution. In physical placement it uses the virtual placement coordinate (with the load dimension equal to zero) to identify a region of the cost space and then select a node (with potentially non-zero load) within that region. As a result operators are only placed on nodes that have sufficient resources to provide services. It is possible that operators are placed sub-optimally in terms of network usage metric. The stage falls into five steps:

- Find k nodes whose coordinates are near the target cost space coordinate.
- Contact this small set of nodes directly to discover their current operators and resources.
- Sort the list by distance to the target coordinate.
- Walk the sorted list, returning the first node already running the operator.
- Failing that, return the nearest node that meets the applications resource criteria.

Figure 2 shows one example of operator placement with above two steps in 2-D latency space. From it we could see that the algorithm favors the centriod of producers and consumers for the unpinned operator, minimizing the potential energy in the supposed spring system.

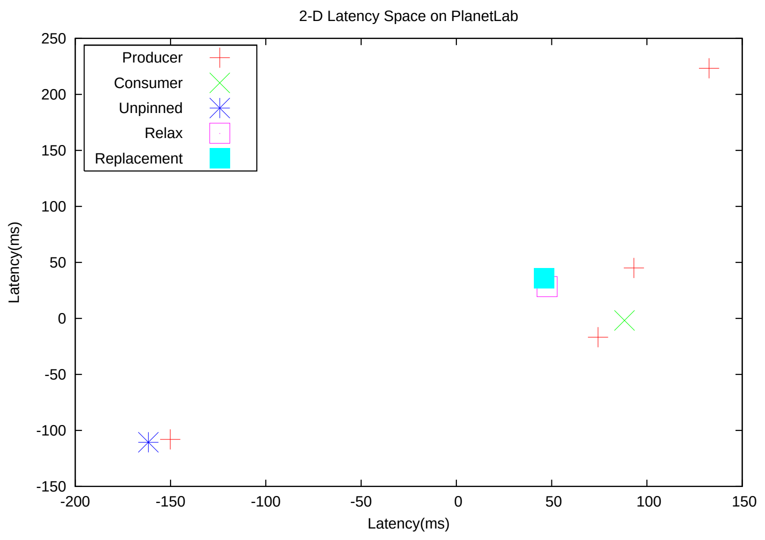


Figure 2: Example of RELAXATION Placement

4 Evaluation

4.1 System Architecture

The key idea of our project is to do all the calculations in a virtual space. Once we get an solution placement, the solution will be port-mapped back into the real physical world. Based on this concept, our project could be divided into the following steps:

- Get the *4 hour PlanetLab ping trace*[6] from Harvard University as input.
- Run *Vivaldi* algorithm on the trace file. The result is presented in a 3-dimensional latency space.
- For each query, use *RELAXATION* algorithm to obtain the optimal operator placement in the latency space.
- For each optimal solution, it will be port-mapped back into the physical network to find a corresponding operator in the real world.
- Result and its evaluation as well as comparison to other operator placement will be presented

4.2 Simulation Setting

Our original plan is to use a network simulator like *NeroGrid*[8] with Georgia Tech topology generator[7] to generate a "controllable" network. Because with a "controllable" network, we expect to be aware of the routing latency between two nodes in the network. However, neither of these simulators generates networks in a "latency-sensitive" way. Alternatively, one segment of trace(4 hour PlanetLab ping trace) could be obtained from [6], which contains 226 nodes in PlanetLab. We could evaluate the performance of operator placement algorithms on this specific network if we do not consider the effect of network topology on placement algorithms.

The *Vivaldi* algorithm will return a 3-dimensional latency space based on the trace file. Note that the load dimension is not represented in this 3-dimensional coordinates. We believe that it is relatively easier to compare the load information on a node compared to that of the latency, because it could be done by simply sorting all the candidate nodes and selecting the one with the best load capacity. On the other hand, latency information always involves in more than just one node on the routing path, therefore is more complicated to represent. Our result for the cost space based on the trace file is demonstrated in Figure 3:

Producers and consumers were randomly distributed across the network. Only a single producer operator was hosted at any physical node. A query consisted of four producers, an intermediary join and select operator, and a consumer. We refer to the join and select as a single

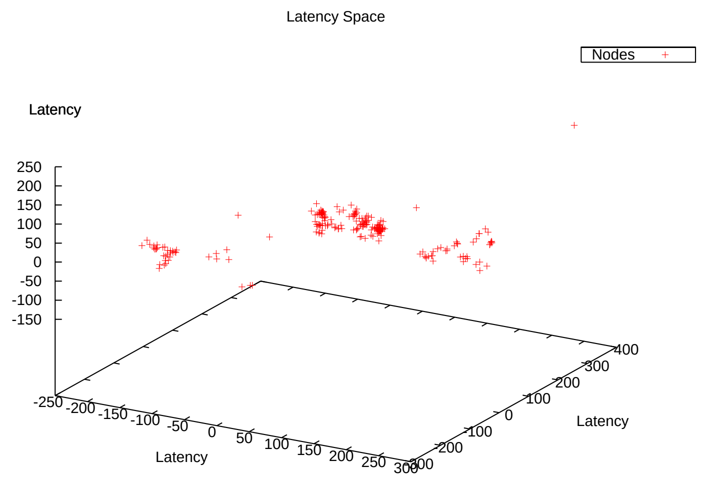


Figure 3: Latency Space

aggregator operator. All producers sent data at a rate of 2KB/s and the aggregator had a selectivity of 4:1.

We examined four other placement approaches: OPTIMAL chooses the best possible placement with the lowest network usage based on an exhaustive search over all possible placements. Although it is not feasible in practice, it gives a baseline for the performance of the algorithms. PRODUCER picks one of the nodes hosting producers for placement of the unpinned operator and averages results of all producers. CONSUMER places the operator at the node hosting the consumer. Finally, RANDOM picks a physical node for each operator at random and serves as a worst case comparison.

4.3 Placement Efficiency in Simulation

The average increase in network usage after placing 1000 random queries is shown in Table 1. We could see that the OPTIMAL minimizes this placement metric. RELAXATION as expected performs very well. PRODUCER averagely does better than CONSUMER because data from only three producers needs to be sent through the network to the operator. RANDOM has the worst performance among five algorithms. The distribution of network usage over all queries is shown in Figure 4.

As we can see in Figure 4, OPTIMAL receive the best network usage feature, followed by RELAXATION. Both PRODUCER and CONSUMER perform much better than RANDOM, while the former is slightly better than the latter. This can be explained by the set up of our query. If we place the unpinned operator in one of the four producers, only three producers left have to send data to the operator through the network.

Delay penalty is the increase in delay when compared to the IP routing delay on the longest

Algorithms	Network Usage	Delay Penalty
OPTIMAL	0%	1.01%
RELAXATION	1.34%	7.71%
PRODUCER	24.91%	50.99%
CONSUMER	27.36%	0%
RANDOM	54.14%	57.26%

Table 1: Increase in Network Usage and Delay Penalty

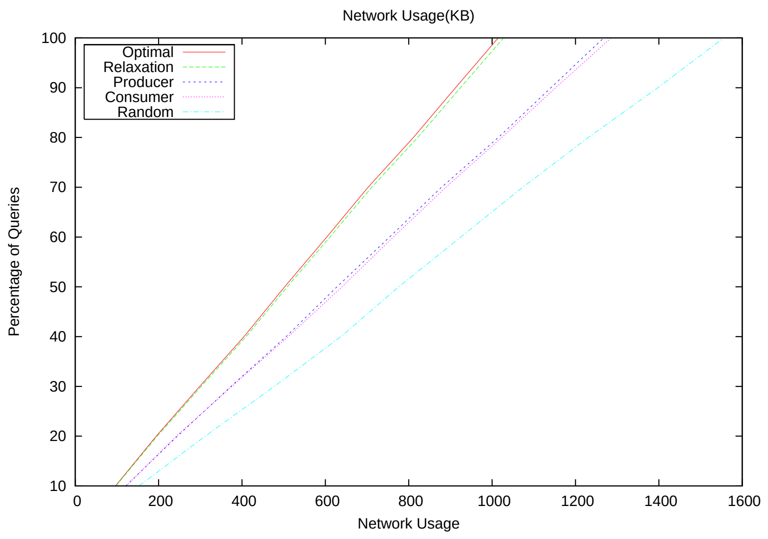


Figure 4: Network Usage

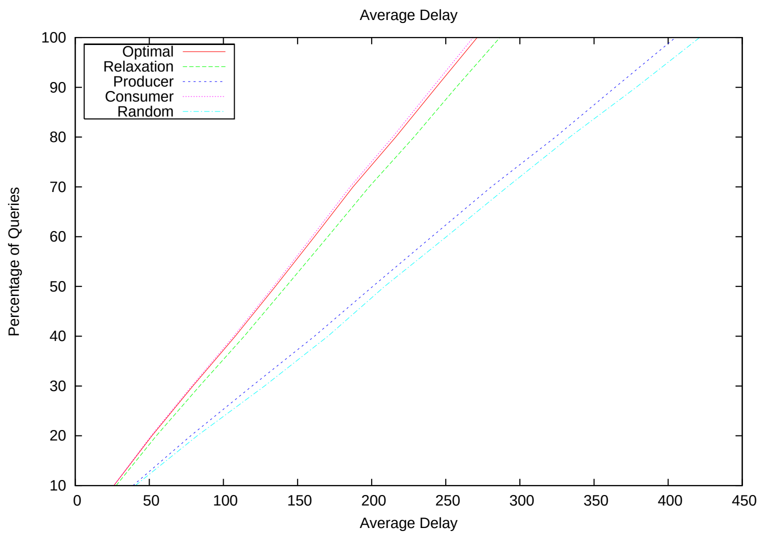


Figure 5: Average Delay

path from any producer to the consumer[1]. It is apparent that placing all unpinned operators on the consumer node achieves the lowest delay penalty because the delay is the longest path from one of the 4 producers to the consumer. Table 1 shows us the average delay penalty after placing 1000 queries.

5 Future Work

It is likely that more performance parameters such as bandwidth and node reliability could added to build up the cost space. The evaluation function should be modified accordingly. For simplicity reasons, we consider query latency only in our project. And we only implemented the full placement optimizer to place operators initially. The placements should be periodically reevaluated by local placement optimizers and migrations of operators could be initiated when necessary.

6 Conclusion

Through software simulation we shows that the SBON creates data streams that minimize overall network usage, a characteristic that is increasingly important as the number of data streams in the network increases. The SBON could do so with an intelligent operator placement, based on the characteristics of the stream and a new placement metric. The cost space of overlay network enables nodes to make adaptive placement decisions with local information and spring relaxation placement algorithm makes efficient operator placement decisions within the cost space.

References

[1] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer,*Network-Aware Operator Placement for Stream-Processing Systems*, Proc. of IEEE ICDE, Atlanta, GA, April 2006.

[2] M. Cherniack, H. Balakrishnan, M. Balazinska, et al.*Scalable Distributed Stream Processing*, Proc. of CIDR, Asilomar, CA, Jan. 2003.

[3] Y. Drougas, T. Repantis, V. Kalogeraki, *Load Balancing Techniques for Distributed Stream Processing Applications in Overlay Environments,* , Proc. of 9th IEEE ISORC 2006, Gyeongju, Korea, April 2006.

[4] Y. Xing, S. Zdonkic, J-H Hwang, *Dynamic Load Distribution in the Borealis Stream Processor*, Proc. of ICDE 2005.

[5] Peter Pietzuch, Jonathan Ledlie, and Margo Seltzer, `Supporting Network Coordinates on PlanetLab`, In Proceedings of WORLDS 2005, San Francisco, CA, December 2005

[6] http://www.eecs.harvard.edu/ syrah/nc/

[7] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, *How to Model an Internetwork*, In Proc. of INFOCOM, San Francisco, CA, Mar. 1996.

[8] http://www.neurogrid.net/php/simulation.php