Network Awareness in Internet-Scale Stream Processing

Yanif Ahmad, Uğur Çetintemel
John Jannotti, Alexander Zgolinski, Stan Zdonik
{yna,ugur,jj,amz,sbz}@cs.brown.edu
Dept. of Computer Science, Brown University, Providence, RI 02912

Abstract

Efficient query processing across a wide-area network requires network awareness*, i.e., tracking and leveraging knowledge of network characteristics when making optimization decisions. This paper summarizes our work on network-aware query processing techniques for widely-distributed, large-scale stream-processing applications. We first discuss the* operator placement *problem (i.e., deciding where to execute the operators of a query plan) and present results, based on a prototype deployment on the PlanetLab network testbed, that quantify the benefits of network awareness. We then present a summary of our present focus on the* operator distribution *problem, which involves parallelizing the evaluation of a single operator in a networked setting.*

1 Introduction

As applications involving large numbers of geographically distributed data sources proliferate, network efficiency and scalability are emerging as key design goals for future data processing systems (e.g., [6, 4, 7, 9, 11]). Central to achieving these goals is *network awareness*, i.e., the capability to track and exploit information about specific network characteristics such as inter-site latencies, network topologies and link bandwidths. Traditional query optimization models are commonly designed for CPU- and disk-bound optimization and thus need to be rethought for network-bound processing.

To this end, the SAND project at Brown is currently developing a network-aware distributed query optimization framework in the context of Internet-scale stream processing applications and the Borealis project [1], which strives to develop a full-fledged distributed stream processing engine. In this paper, we provide an overview of SAND, focusing on two key problems that arise in networked stream processing:

- *Operator placement* — deciding the network locations where the operators of a query plan should be placed and executed.
- *Operator distribution* — deciding how a single operator, such as a join, should be distributed across nodes in a networked setting.

Even though both problems have been extensively studied in the context of traditional distributed and parallel databases (e.g., [10, 5]), existing solutions typically assume a small-scale, static and homogeneous networking

This work has been supported in part by the National Science Foundation under the ITR grant IIS-0325838.

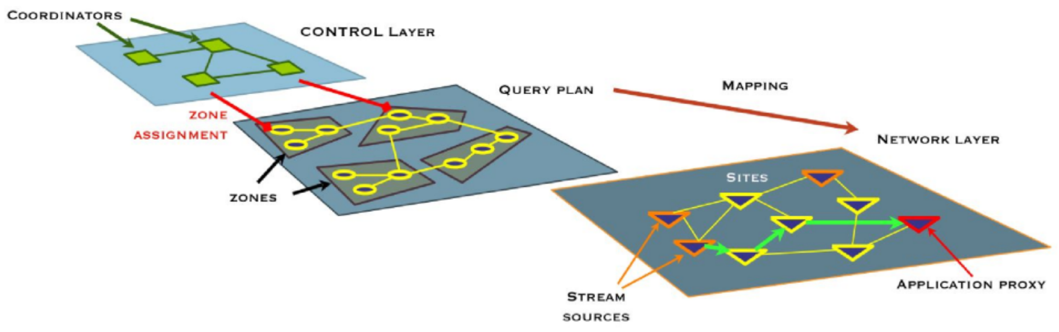


Figure 1: A query plan and the corresponding control and network layers.

environment and rely on centralized algorithms. These solutions, thus, fall short of addressing the unique requirements of Internet-scale query processing where network awareness and distributed algorithms are crucial for achieving efficiency and scalability. Our work strives to eliminate these assumptions and limitations of prior work.

We first summarize SAND's basic system model. We then present various operator placement algorithms that differ primarily in what nodes they consider as candidates for operator placement and how they take network knowledge into account. We follow by presenting experimental results that characterize the relative performance of the approaches in a wide-area setting, through deployment on the PlanetLab network testbed on top of the Borealis stream processing engine. Finally, we briefly discuss the operator distribution problem and highlight the key ideas that we are currently exploring towards an efficient and scalable solution.

2 Networked Operator Placement

2.1 Basic System Model

We target a widely-distributed query processing environment with geographically dispersed data sources that produce high-volume data streams that are to be processed for purposes of filtering, fusion, aggregation, and correlation. We assume an underlying distributed stream processing system (such as Borealis) that provides core stream processing functionality and mechanisms for dynamic operator migration across nodes.

We abstractly represent our query plan as a tree of operators, called *a processing tree*, whose edges represent outputs from one operator that are used as inputs by another operator. The query's data sources are represented as leaves of this tree. We use a cost function combining (1) the bandwidth used while transporting data between operators and (2) the network latency between the nodes selected to host the operators. The bandwidth usage here is obtained by considering input rates to the operator and the operator's selectivity. Two connected operators incur a cost of zero if they are located at the same node, capturing the property that no network cost is incurred if sequential operators are evaluated locally.

We use three heuristics-based algorithms to construct a placement minimizing this cost function: (1) *Edge*, a network-agnostic algorithm placing operators at the data sources alone, (2) *Edge+*, an extended version of Edge that factors in network latencies and topology during placement, and (3) *In-Network*, an algorithm capable of placing operators at arbitrary locations within the network.

2.2 Control Model

For improved scalability and parallelism, we use a distributed control model. For each processing tree, we create a corresponding *control tree* of coordinator nodes by logically partitioning the processing tree into a number of subtrees (not necessarily extending to the leaves), called *zones*. Each zone is then assigned a coordinator node that is responsible for the placement (and periodic dynamic re-placement) of its operators. The *application proxy*, the node that delivers the output of processing to the clients, is always assigned as the root coordinator and decides how many additional zones to create.

Figure 1 provides a high-level view of this model, illustrating a processing tree as well as the corresponding control and processing networks overlayed on top of the IP network. The processing tree is partitioned into three zones, each assigned to a coordinator node. Each coordinator maps all operators in its zone using one of the aforementioned heuristics. An advantage of this distributed optimization model is that zones can be optimized locally, concurrently and asynchronously.

Note that a coordinator is dependent upon other coordinators whose zones contain descendant operators. Our operator tree is, thus, mapped in a bottom-up fashion, starting at coordinators responsible for leaves, and triggering placement of parent operators as each individual operator is mapped. Coordinators communicate with each other when one coordinator has mapped its zone, sending optimization metadata to the coordinator responsible for mapping parents. The contents of this metadata depend on the heuristic in use and will be described along with the heuristics. In our distributed algorithm, a coordinator may also backtrack on the placements prior to its round of control. This approach helps to overcome locally optimal solutions that are not necessarily globally optimal, given operators are initially mapped without knowledge of their siblings residing in other zones.

2.3 Placement Algorithms

The algorithms in this section determine an operator's placement, given the placement of the operator's immediate children. The heuristics also determine if backtracking is necessary, and provide updated positions for the operator's children.

- The *Edge* heuristic considers placing an operator at the locations its children are placed, and any *common* locations. We define common locations as follows. Each data source (i.e., the leaves of our tree) is assumed to reside at a fixed location in the network. An operator may potentially be placed at the locations of any data sources providing its inputs. A common location is a node present in the set of potential locations of each child. As previously mentioned, Edge is a network-agnostic heuristic that uses a cost function of bandwidth alone, and does not consider network latencies between nodes. Edge simply places the operator at the location minimizing the total cost of each child's subtree, and any edges between the operator and its children. If the selected location is a common location, rather than some child's existing location, each child's location is updated to the common location. When using this heuristic, coordinators must send the set of potential locations for the root of their workload, as well as the cost of placing this root at each potential location. The recipient coordinator may then map a parent, attempting to determine an optimal solution by reconsidering each child's placement in light of their respective placements.

- The *Edge+* heuristic incorporates network latencies between nodes when making its placement decision. Edge+ considers placing operators at their child and common locations, in an identical manner to Edge. However, rather than considering cost as the bandwidth utilized by parent-child edges alone, Edge+ calculates a cost based on the product of edge bandwidth and latencies. The latency here is that between a potential placement of the operator, and the placement of a child. Edge+ searches over all potential placements of the both the operator and its children, selecting a configuration with the minimal bandwidth-latency product. In addition to an expanded cost function, Edge+ also incorporates a pruning technique not found in Edge. Specifically, Edge+ prunes potential placements of an operator based on a latency criterion. This criterion eliminates locations if the total latency between the location under consideration and the locations of all children exceeds the total latency between child locations alone. In a similar manner to Edge, coordinators exchange optimization metadata comprising of placement costs at potential locations.

- The *In-Network* heuristic extends Edge+ in the set of potential locations considered for placement, and in the pruning heuristic used to eliminate locations. In-Network searches over a set of candidate locations obtained as a shortest-path tree between the application proxy, and all data sources. We remark that candidate node selection remains an open issue and are investigating several other heuristics (such as candidates chosen by random walks and by directed flooding in the neighborhood of our data sources). In addition to pruning based on a distance criterion as in Edge+, In-Network computes a ranking based on the total latency between potential operator locations and child locations, and selects the top k such locations to actually compare costs (k is a configuration parameter). Consequently, coordinators only forward optimization metadata containing placement costs at this subset of potential locations.

We refer the reader to [2] for a detailed description of the placement algorithms, as well as extensions that can meet user-specified per-query latency bounds.

3 System Architecture and Deployment

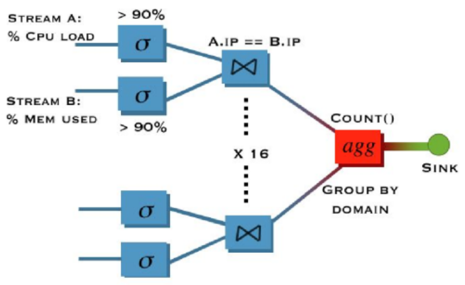
We have implemented a SAND prototype using the OCaml language, and deployed our implementation across the PlanetLab testbed. Our key design principles included retaining a flexible optimization and network protocol framework, allowing modular extensibility to experiment with various operator placement heuristics. To support this extensibility, our prototype includes a loosely coupled protocol stack and optimization engine. Lower layers of our protocol stack include a DHT-driven lookup protocol (using the OpenHash implementation) and latency and bandwidth probing protocols.

The SAND prototype interfaces with the Borealis stream processing engine, and uses this engine to perform widely distributed query processing. Borealis provides an XML-RPC interface enabling SAND to both collect selectivity statistics from a running query, and subsequently perform operator placement by moving Borealis boxes. Presently the two systems are loosely coupled, with SAND capable of mapping a Borealis query network onto a physical network in an offline manner. Our recent efforts have been to deploy the Borealis prototype onto PlanetLab, implementing a wide-area network monitoring tool as an example application. This application generates input data for Borealis through the use of a data wrapper interacting with the Ganglia, Slicestat and Comon "sensors" collecting statistics from each PlanetLab site, and periodically pushing tuples into the Borealis engine running at that site. We expect to integrate the SAND and Borealis prototypes to collect online selectivity estimates in the immediate future.

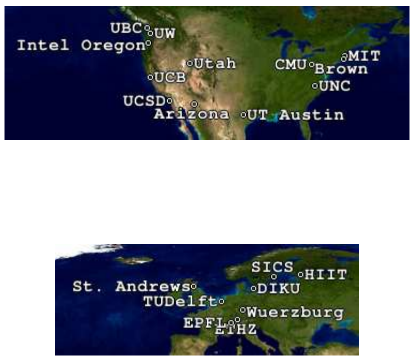
4 Experimental Results

We briefly present two sets of experiments, where SAND first maps abstract query plans containing a variety of edge rates between operators, and secondly maps a Borealis query network prior to measuring actual edge rates on the running query. Both sets of experiments were run on Planet Lab.

Our experimental evaluation methodology compares the bandwidth consumption ratio and stretch factor of the mapping obtained to a "warehousing" scenario, where data sources push tuples to a single site in the network and the query is evaluated entirely at this site. The bandwidth consumption ratio is defined as the total sum of our bandwidth-latency product of all transfers between operators across our network, as a ratio of the equivalent metric in the centralized scenario. The stretch factor is a ratio of the maximum end-to-end latency resulting from our mapping compared to the centralized scenario.



i)



ii)

Figure 2: Experiment setup and deployment across Planet Lab

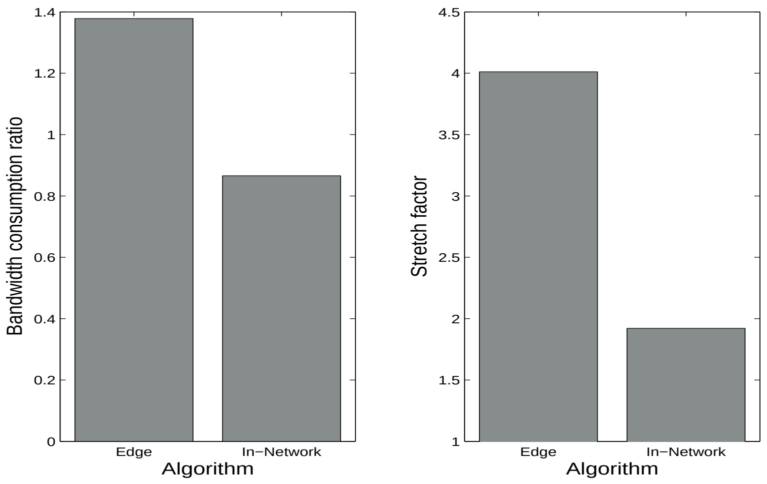


Figure 3: i) Bandwidth consumption ratio and ii) Stretch factor of abstract query plan mappings.

Figure 2 shows the Borealis query plan we deployed across 20 PlanetLab sites chosen on the East and West coasts of North America and Europe. Each data source provides statistics on CPU and memory utilization. The query identifies the sites with very high resource utilization and outputs an aggregate count of those, grouped by their domain identifiers. In the final placement, a filter and join box are placed at each source site, and the aggregate box is mapped to the MIT site. We collect the query's results at Brown.

Our abstract representations of query plans were binary trees of depth four, and fanout two. Our results in Figures 3i) and 3ii) confirm those witnessed in our emulation [2], namely the dominance of the In-Network heuristic over the network-agnostic Edge heuristic both in terms of the bandwidth consumption ratio, and the stretch factor. Indeed the incorporation of network awareness is key: results comparing Edge and Edge+ (omitted) exhibit similar gains in the Edge+ heuristic.

Figure 4 shows the bandwidth consumption ratio and stretch factor measured for the query network in Figure 2, for two different selectivity values. To achieve these selectivity values, we used a dummy CPU and memory utilization sensor generating values between 0-100%, and subsequently adjusted the threshold of our filter's predicate. The results here again demonstrate the benefits of in-network processing, where in this scenario, the join operators are performed at the source sites, and the aggregate operator at a centroid-like location (in this case MIT) with respect to all sources. The differences in the absolute values of the bandwidth consumption ratio between Figure 3 and Figure 4 arise due the higher selectivity values used in the former set

	Selectivity									
	0.1		0.2		0.3		0.4		0.5	
# sources	BWCR	SF	BWCR	SF	BWCR	SF	BWCR	SF	BWCR	SF
8	0.2154	1.2589	0.2510	1.2723	0.2940	1.2749	0.3382	1.2510	0.3890	1.2642
16	0.2365	1.3469	0.2710	1.4031	0.3227	1.3945	0.3686	1.4052	0.4146	1.4073

(BWCR = bandwidth consumption ratio, SF = stretch factor)

Figure 4: Results for the Borealis processing network from Fig. 2

of experiments.

5 Operator Distribution

As part of ongoing work in the SAND project, we are investigating mechanisms to further the distribution of a query plan across a heterogeneous network. Our current focus is on the parallelization of query operators, motivated by the vast divide between the scale of the abstract processing graph representation of a query, and the abstract graph representation of the network. To this end, we are pursuing techniques enabling an individual operator to utilize multiple sites during evaluation, while considering the necessary interaction between these sites to provide an equivalent operational semantics as would result from a centralized evaluation.

Our distribution mechanisms revolve around *replicating* and *partitioning* operators. To effectively support distribution, we focus on exploiting two properties: *network locality* and *data locality*. Network locality refers to the proximity of the data sources' network locations. Data locality refers to the similarity between data sources in terms of the input values produced, and the frequency at which these values are produced. Data locality also captures temporal properties of the inputs, such as the synchronicities of the input values.

We now discuss these mechanisms in the context of evaluating a join operator in a networked environment, with the goal of improving the average end-to-end latency of processing tuples (more details and relevance to previous work can be found in [3]):

- *Replication*: Our replication mechanism constructs a *join tree* of replicas (i.e., join instances), with each replica capable of performing a partial evaluation of the join operator based on the subset of all data sources it receives inputs from. Following partial evaluation, an operator replica may immediately deliver its output to any interested parties, creating a "short-circuit" route between sources, processing site, and sinks. We rely on a routing and evaluation policy between replicas to ensure correctness in our evaluation. First, operators in our tree evaluate the join predicate on tuples only if they arrive on differing input branches. Second, operator replicas forward input tuples received from their children towards the root of the tree. The first property ensures that our replica tree does not produce duplicate results, allowing input tuples to only join at their sources' lowest common ancestor in the tree. The second property ensures completeness of outputs, since all tuples will be forwarded to the root. This replication mechanism will prove beneficial in terms of average end-to-end latency provided the majority of the join's output occurs at the lower levels of the replica tree and thus a decreasing join selectivity at operator replicas closer to the root. We note that PIER recently considered a technique that performs a similar hierarchical join but in a rather opportunistic manner [8].

- *Partitioning*: While replication targets exploiting network locality amongst data sources by enabling nearby sources to compute their join results as early as possible, a second mechanism, operator partitioning, focuses on improving the networked deployment of a join operator by exploiting data locality amongst sources. Operator partitioning requires sources to introspectively maintain a probability distribution over the input values they produce. Sources subsequently exchange these distributions, using customizable approximate representations such as wavelet-based histograms, to compute join output probability distributions. A distributed partitioning algorithm identifies trends in these distributions across pair-wise combinations of data sources, placing partition boundaries on the join-key attribute domain according to a gradient-based heuristic applied to the output probability distribution. In this way, partitions are chosen to highly differentiate the set of sources that produce output values for each input value contained within each partition.

Note that the replication and partitioning mechanisms are orthogonal in their applicability, and thus may be composed: we can create a replica tree for each partition instantiated, first applying our distributed partitioning algorithm before constructing the replica tree using a hierarchical clustering algorithm.

5.1 Implementation Plans

Our next phase of implementation involves integrating these proposed algorithms into the SAND framework, and investigating the benefits these algorithms would bring to both a massively multiplayer online game, while performing "area-of-interest"-based data dissemination, and our wide-area network monitoring tool deployed on PlanetLab on top of the Borealis stream processing system.

References

[1] D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Jan. 2005.

[2] Y. Ahmad and U. Cetintemel. Network-Aware Query Processing for Distributed Stream-Based Applications. In *Proc. of the 30th International Conference on Very Large Data Bases (VLDB'04)*, 2004.

[3] Y. Ahmad, U. Cetintemel, J. Jannotti, and A. Zgolinski. Locality Aware Networked Join Evaluation. In *Proc. of the 1st International Workshop on Networking Meets Databases (NetDB'05)*, 2005.

[4] S. Chandrasekaran, A. Deshpande, M. Franklin, and J. Hellerstein. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. of the First Biennial Conference on Innovative Data Systems Research (CIDR'03)*, Jan. 2003.

[5] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsaio, and R. Rasmussen. The Gamma Database Machine Project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990.

[6] M. Franklin, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design Considerations for High Fan-in Systems: The HiFi Approach. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Jan. 2005.

[7] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4):22–33, Oct. 2003.

[8] R. Huebsch, B. Chun, J. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of PIER: an Internet-scale query processor. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Jan. 2005.

[9] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proc. of the 29th International Conference on Very Large Data Bases (VLDB'03)*, Sept. 2003.

[10] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[11] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: An Acqusitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems*, 2005.