Throughput-Optimized, Global-Scale Join Processing in Scientific Federations

Xiaodan Wang, Randal Burns, Andreas Terzis
Computer Science Department
The Johns Hopkins University
{xwang, randal, terzis}@cs.jhu.edu

Abstract

We present distributed query scheduling algorithms that minimize network utilization for spatial joins in the SkyQuery federation of Astronomy databases. Unlike existing works that measure the quality of join schedules based on query response time, our metric both minimizes network utilization and balances the utilization of heterogeneous network paths. Preliminary experiments show that our algorithms reduce network utilization dramatically when compared with SkyQuery's existing scheduling algorithm.

1 Introduction

Making observational data available for analysis over the Internet is increasingly important in the disciplines such as Astronomy and Biology. The SkyQuery federation of Astronomy databases [13] provides one such service. SkyQuery brings together multi-terabyte, Astronomy archives that catalog and map spectral characteristics of celestial objects. However, SkyQuery faces an impending scalability crisis both in terms of the number of sites and data size. The federation is expected to expand from a few dozen members today to over a hundred in the near future [22].

Data volume and geography deter scalability. The workload in SkyQuery is network-bound. It is not uncommon for join queries to yield intermediate results that are hundreds of megabytes or more in size [12]. Also, the physical distribution and connectivity of sites in SkyQuery vary tremendously with roughly 30 sites distributed across North America, Western Europe, and East Asia. In this heterogeneous environment, transferring data between sites that cross continental boundaries is highly undesirable, because paths that cross long physical distances exhibit higher propagation delay, congestion, and loss rates. Query processing should exploit network locality and avoid narrow, long-haul paths by making these paths more costly during optimization.

Existing algorithms either reduce the volume of network traffic produced or minimize the time to complete join queries across heterogeneous networks, but do not ensure efficient allocation of network resources. Specifically, algorithms that focus on reducing the volume of traffic under-utilize the network because paths with excess capacity may be overlooked. Algorithms that minimize completion time over-utilize the network by consuming all available resources to achieve a locally optimal plan. Our work differs in that we minimize network utilization balanced over all network paths, using a single metric.

We describe two polynomial-time algorithms, including a spanning tree algorithm with provable bounds and a tunable, clustering-based heuristic for balancing network and computation cost. The cost model we use rewards join schedules that utilize paths with excess capacity and produce small intermediate join results. Our experiments show that our algorithms outperform SkyQuery's existing scheduling algorithm and scale to large joins. Furthermore, the network performance of our algorithms tracks closely with the optimal query schedule, determined by an exhaustive search of all feasible schedules.

2 Query Scheduling in SkyQuery

The SkyQuery [13] federation of Astronomy databases makes data available to the public through Web-services and Web-forms. Users submit queries through a mediator, which communicates with member databases via a shared wrapper interface. The federation exhibits a large amount of network heterogeneity; geographically collocated sites form highly-connected clusters, whereas inter-cluster connectivity is weak. The federation supports both standard SQL queries against a single site and spatial join queries involving an arbitrary number of sites.

Cross-match [13], the principal query in SkyQuery, joins observations of the same astronomical object from several databases by correlating their location in space. Cross-match extends SQL by adding two clauses. The `region` clause specifies an area for conducting the search. The `xmatch` clause specifies an unordered list of databases to visit. In Figure 1, we show the execution of a sample cross-match query involving three sites.

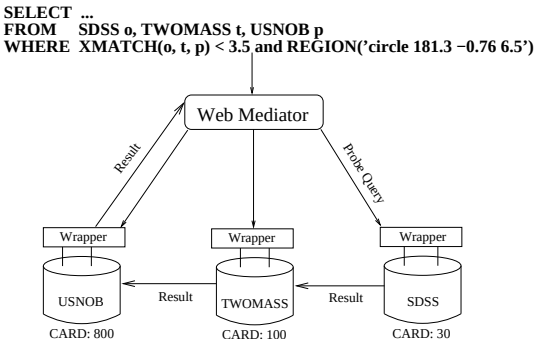

Figure 1: Probabilistic spatial joins in SkyQuery

SkyQuery employs a *count* $*$ [13] approach to produce join schedules for cross-match queries. In *count* $*$, the mediator first issues probe queries to all sites participating in the join. Probe queries determine site cardinality, defined as the number of rows that satisfy the `region` clause from each site. This is accomplished by executing a SQL `select count(*)` version of the user query. Based on the probe queries, the mediator produces a serial schedule joining sites in ascending cardinality order.

The *count* $*$ approach is effective at early elimination of tuples that do not participate in the join at subsequent sites. This approach minimizes computation for databases on uniform networks under a couple of assumptions: uniform join probabilities and linear I/O and processing costs in the number of tuples. Instrumenting SkyQuery shows that these assumptions hold. However, the insensitivity of *count* $*$ to geography produces poor schedules in a global-scale federation. Nothing prevents *count* $*$ from choosing a narrow network path over a high capacity path.

3 Related Work

Most commercial optimizers use algorithms based on dynamic programming [9]. System R-style dynamic programming algorithms [4, 20] guarantee an optimal solution but have exponential-time complexity. Polynomial-time dynamic programming algorithms [9, 11] are more attractive because they provide near-optimal schedules and scale to large joins. However, dynamic programming cannot be used when optimizing balanced network utilization, because this metric lacks optimal sub-problem structure. For example, an optimal relative order over three sites may no longer be optimal once a fourth site is introduced.

Several works optimize join queries by minimizing the size of intermediate results [11, 19]. Lee proposed a graph model [11] that represents join operations as nodes and the execution order as directed edges. Join schedules are found using a polynomial-time algorithm, which identifies the minimum cost spanning tree in a graph. These algorithms reduce computation and network costs on uniform networks but do not account for network heterogeneity.

Richer formulations of join scheduling reduce response time based on the joint optimization of computation and network costs. Mermaid [1], an early testbed for multidatabase systems, models both transmission cost and execution cost. Evrendilek [5] studies the problem in the presence of data replication and accounts for the execution time of queries at runtime for load balancing. These algorithms greatly expand the search space by considering bushy join plans, which exploit parallelism. However, minimizing the response time of join queries through parallel execution sacrifices job throughput and leads to unbalanced resource allocation [6].

Data-centric routing in wireless sensor networks shares some techniques with our work. Several works study the problem of routing data from multiple sensors to a single base station in which in-network consolidation of data is allowed [8, 10, 21]. Some of these schemes organize sensors into a spanning tree rooted at the base station and route data along the tree [10, 21]. Meliou *et al.* [14] study the NP-hard problem of finding the optimal route for gathering data from a subset of sensors. However, the goal in sensor networks is quite different: conserving power by minimizing network usage.

4 Balanced Network Utilization

Our goal in query scheduling is to maximize query throughput in the federation. Achieving this goal often delays the completion time of an individual query. However, Astronomy workloads are not latency sensitive due to the large data sizes. Just accessing the data at each site often takes tens of seconds.

The capacity of a network path measures the impact of geography on throughput. It describes the number of (intermediate) query results that can be sent across a path over a given time frame. Balanced usage of capacity on all paths maximizes the total query throughput. We define a path's capacity as the TCP throughput achievable on a single connection. A TCP-specific performance metric is appropriate because TCP carries nearly 90% of the Internet traffic [18].

In contrast to latency-based distance [16], TCP throughput is non-metric. This matters because many scheduling optimizations rely on the triangle inequality. Our approach is to identify situations in which the triangle inequality holds and use metric optimizations in these regions.

TCP throughput captures geography fairly well. Paths that traverse longer distances exhibit lower TCP throughput [17]. In Figure 2, we plot geographical distance against TCP throughput for 400 node pairs from PlanetLab [3]. The throughput measurements are obtained over a one-hour period using the *netperf* benchmark [15] in which we take the average of three measurements for each pair of nodes. For each site in SkyQuery, we choose a PlanetLab site that is close (in the network) to that server. Thus, network measurements across PlanetLab sites approximate the

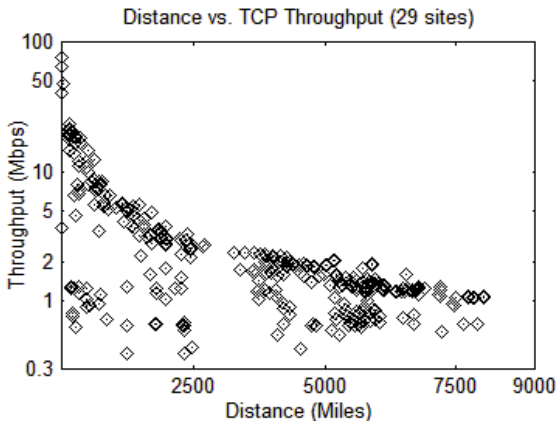

Figure 2: Geographical distance vs TCP throughput

same measures across SkyQuery sites. The results show a significant degree of network heterogeneity. However, geographically collocated nodes often form clusters of well-connected, high throughput sites. For instance, paths that achieve TCP throughput of 10 Mbps or higher are only found in sites less than 600 miles apart.

The path capacity between each pair of sites is obtained by measuring average throughput for large file transfer over TCP. Because measurements are intrusive due to the amount of traffic they generate, paths are not sampled frequently. Paths can also be measured by taking frequent, short-duration samples. However, the latter method exhibits high variability due to the effects of cross-traffic [18] and is an unreliable performance predictor for the transfer of large query results. Average TCP throughput exhibits significantly less variation over time. The most volatile paths never deviate more that 30% from their mean value.

We create a framework for query optimization based on the utilization of network paths. The cost of using a path is the product of the volume of data transmitted and the inverse of its TCP throughput. Thus, we weight each path using the inverse of TCP throughput, which measures the time cost of transmitting data (per byte) on that path. Sending 100 KB over a 1 millisecond per KB (ms/KB) path costs the same as sending 10 KB over a 10 ms/KB path. We define the *balanced network utilization* metric by summing the costs over all paths used in a schedule over n sites:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{VOL_{ij}}{TP_{ij}}.$$

in which TP_{ij} and VOL_{ij} describe the TCP throughput and volume of data transmitted on the path connecting sites i and j. The goal of scheduling is to minimize this metric. This framework benefits scalability in two ways. First, joins are scheduled based on minimum aggregate cost, which alleviates contention on the network. Second, schedules exploit network heterogeneity by choosing paths with high TCP throughput.

4.1 Assumptions and Limitations

Ideally, the optimizer can predict network traffic exactly, based on perfect knowledge of the size of intermediate join results. However, the network and storage costs of gathering and maintaining join statistics of attributes across globally-distributed databases are prohibitively expensive. Maintaining join statistics for every join permutation is impractical even for a dozen sites.

Rather than estimate the selectivity of arbitrary n-way joins, we take a conservative approach and assume perfect join selectivity in which the intersection of an r_m-tuple, r_n-tuple, and r_o-tuple relation produces a result with $min(r_m, r_n, r_o)$ tuples. As a consequence, we may miss sites with low join selectivity that benefit from early evaluation in the join schedule. For Astronomy queries, perfect join selectivity holds surprising well. In queries involving four sites, the cardinality of the join result deviates less than 20% on average from that of the minimum cardinality site.

Similar to the *count* $*$ approach, we ignore the effects of attribute aggregation; that is, each site contributes a non-trivial number of attributes to the intermediate join result. Aggregating attribute values in this fashion means that the join result size may grow significantly, potentially making it more costly to transfer data from a low cardinality site to a high cardinality site compared to the reverse direction. Fortunately, our results show that this limitation does not significantly impact the quality of the solution, because spatial joins in SkyQuery involve few attributes and the number of rows in the result dominate data transfers.

4.2 Sample Join Schedule

Figure 3 shows a sample join schedule involving three sites, S_1, S_2, and S_3 plus the Web mediator or sink, S_0. The bold lines represent the paths used by the join schedule and the dotted lines represent available network paths. Let r_i be the number of rows from S_i that fall inside the query region and w_i describe the number of attributes site S_i contributes to the intermediate join result. Lastly, $d(i, j)$ denotes the inverse of TCP throughput for the path connecting S_i and S_j. To simplify the example, we assume that all attribute values are one byte in size. Sites S_1,S_2, and S_3 contain 100, 10, and 20 rows inside the query region and contributes 3, 1, and 1 attributes to the join result respectively.

The join schedule begins at S_3, which ships all rows that satisfy the `region` clause to S_2. The cost of utilizing the path from S_3 to S_2 is 5 milliseconds (ms) per byte and the total cost incurred on the path is $r_3 * w_3 * d_{32}$ or 100 ms. After joining tuples from S_2 and S_3, the intermediate result contains $min(r_2, r_3)$ rows and $(w_2 + w_3)$ attributes. Thus, the total cost incurred on the path from S_2 to S_1 is $min(r_2, r_3) * (w_2 + w_3) * d_{21}$ or 200 ms. Finally, the cost

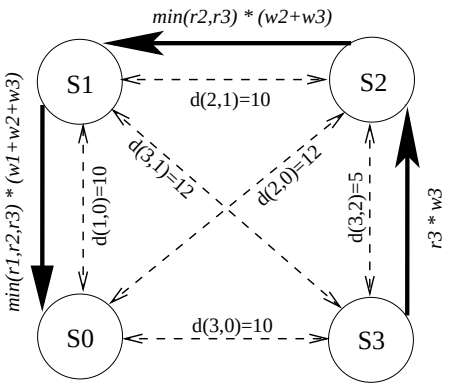

Figure 3: Join schedule with three sites and a sink

on the edge from S_1 to the sink is $min(r_1, r_2, r_3) * (w_1 + w_2 + w_3) * d_{10}$ or 500 ms. Summing the costs over all paths yield a balanced network utilization of 800 ms.

5 Scheduling Algorithms

We describe two scheduling algorithms that use our balanced network utilization metric. The first adapts the two-approximate solution to the traveling salesman problem (TSP) to query scheduling. The second extends this algorithm by clustering the sites based on pair-wise path capacity and then uses the *count* $*$ approach within the clusters and the first algorithm among the clusters. This optimizes computation and I/O in well-connected network regions and balanced network utilization over narrow paths.

5.1 Spanning Tree Approximation (STA)

We achieve a two-approximation of the optimal join schedule by adapting the two-approximate solution to metric TSP [2], which uses a minimum spanning tree (MST). Unlike TSP, a join schedule can visit each site more than once and traverse each network path on the MST at most twice. Our algorithm takes the minimum cardinality site and convolves its data among all sites, following paths in the MST. More formally, provided n join sites, let r_{min} be $min(r_1, ..., r_n)$ and S_{min} correspond to the minimum cardinality site. Given a fully connected graph consisting of the join sites in which the edge weights are the product of the inverse of TCP throughput and r_{min}, compute the MST. Pick S_0 as the root of the spanning tree and recursively enumerate a breadth-first order that visits all nodes in each of the subtrees rooted at its children. These serve as join schedules for each subtree. Finally, combine the join schedules so that the final schedule starts at S_{min}, finishes at S_0, and visits each node at least once.

The network cost of the resulting plan is at most twice the sum of edge cost of the minimum spanning tree, $cost(MST)$, because each edge is traversed at most twice: once to visit all descendant nodes and once to return to the parent. The edges connecting S_{min} to S_0 are traversed only once. In addition, the network cost of the optimal schedule, $cost(OPT)$, is at least as large as $cost(MST)$.

Table 1: Clustering the TCP Throughput Matrix (Mbps)

	US1	US2	US3	US4	US5	US6	US7	EU1	EU2	EU3
US1	**100**	**47.2**	2.68	2.91	2.89	2.97	2.96	1.3	1.26	1.28
US2	**47.2**	**100**	2.67	2.79	2.24	2.95	2.95	1.31	1.25	1.28
US3	2.68	2.67	**100**	**18.0**	**16.7**	**14.5**	**14.6**	2.35	2.07	2.15
US4	2.91	2.79	**18.0**	**100**	**77.4**	**21.5**	**20.8**	2.2	1.58	1.85
US5	2.89	2.24	**16.7**	**77.4**	**100**	**24.2**	**23.5**	2.23	1.54	1.75
US6	2.97	2.95	**14.5**	**21.5**	**24.2**	**100**	**81.6**	2.25	1.61	1.85
US7	2.96	2.95	**14.6**	**20.8**	**23.5**	**81.6**	**100**	2.24	1.79	2.25
EU1	1.3	1.31	2.35	2.2	2.23	2.25	2.24	**100**	**8.62**	**8.97**
EU2	1.26	1.25	2.07	1.58	1.54	1.61	1.79	**8.62**	**100**	**27.3**
EU3	1.28	1.28	2.15	1.85	1.75	1.85	2.25	**8.97**	**27.3**	**100**

In the two-approximate solution to metric TSP, an Eulerian tour is constructed on the MST that traverses each edge at most once. Since this tour avoids backtracking on the MST by going directly to the next unvisited node, each node is visited exactly once as required by TSP. The triangle inequality ensures that the cost of the direct route is less than the backtracking path on the MST. We observe that the triangle inequality holds frequently, despite the fact that TCP throughput is non-metric. By exploiting metric regions in which the triangle inequality holds and bypassing previously visited sites in the join schedule, we further reduce network cost by up to 30%. However, this does not improve the algorithmic bound.

5.2 Clustered STA (C-STA)

We observe from PlanetLab experiments that sites located in close geographic proximity form well-connected components. Thus, clustering sites based on TCP throughput is possible because the throughput on intra-cluster paths is relatively uniform and consistently high when compared with the throughput on inter-cluster paths.

We employ the bond energy algorithm (BEA) [7] for clustering SkyQuery sites. BEA is a heuristic algorithm with complexity $O(n^2)$ that operates on object-object data arrays and reorders the rows and columns in order to maximize the similarity between the data values of neighboring entries. We use the TCP throughput between every site pair to populate the data array and then reorder the array using BEA such that the similarity in TCP throughput among neighboring entries is maximized. Finally, groups of related nodes are extracted from the array based on a threshold. Table 1 illustrates the result of applying BEA on the TCP throughput data array for a subset of nodes: 10 sites distributed across North America and Europe. This example forms three clusters in bold, using a threshold of 3.0 Mbps. The minimum value inside clusters exceeds the maximum value outside of clusters. Running BEA on all sites with a threshold of 3.0 Mbps results in six clusters for the 30 PlanetLab proxy nodes.

The clustered spanning tree approximation (C-STA) uses site clusters to hierarchically combine *count* $*$ with the

spanning tree approximation (STA). First, the algorithm selects a join order within each cluster using *count* ∗. Then, it runs STA on the intermediate results output from each cluster.

C-STA minimizes both computation cost and balanced network utilization. It inherits an important benefit of the *count* ∗ approach. Inside each cluster, the join order minimizes the size of intermediate results, which lowers computation cost relative to STA. Among clusters, on the low-capacity paths, the algorithm minimizes balanced network utilization. The relative importance of computation and network utilization depends on the selection of the threshold parameter. In fact, the threshold can be tuned to balance network utilization and computation cost without having to solve a multi-objective optimization problem. For example, with an extremely low threshold, all sites are grouped into a single cluster and the algorithm devolves into *count* ∗. With a high threshold, each site forms its own cluster and the algorithm is equivalent to the spanning tree approximation. Tuning the threshold offers a continuous family of algorithms that explore computation/network trade-offs. Currently, we have no automated way to perform parameter selection. Selection requires a relative valuation of computation and I/O versus network utilization, which are not directly comparable.

5.3 Attribute Aggregation Revisited

As mentioned earlier, we ignore attribute aggregation when scheduling join queries. Introducing attribute aggregation is problematic because edge weights are no longer bounded by the minimum cardinality site in STA. Attribute aggregation is addressed using a two-phase approach in which a join schedule is first found through STA or C-STA. Next, permute the join schedule in order to delay joins involving sites that contribute a large number of attributes. Given a join schedule found by STA, we push sites with many attributes further back in the join order until all network cost benefits are exhausted. Our results demonstrate the efficacy of this simple two-phase approach.

6 Experiments

We evaluate the performance of spanning tree approximation (STA), clustered spanning tree approximation (C-STA), and *count* ∗ approaches relative to the optimal join schedule (OPT). OPT is obtained by enumerating all possible left deep join orders and picking the lowest cost plan. OPT accounts for attribute aggregation, but makes the uniform join probabilities assumption. Because *count* ∗ does not consider TCP throughput, network utilization is over ten times worse than OPT in joins involving twelve sites.

Figure 4 shows network utilization normalized to the OPT for queries involving two to twelve sites with each site contributing up to ten attributes to the join result. The results for each join size is averaged from 100 SkyQuery

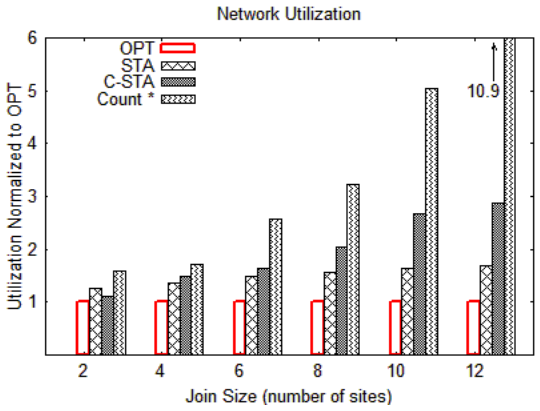

Figure 4: Network utilization normalized to OPT

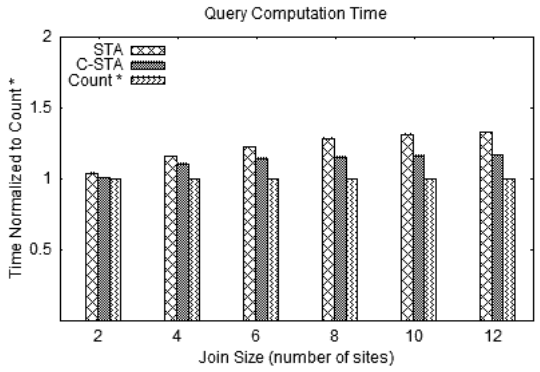

Figure 5: Query computation time normalized to *count* ∗

queries. STA is clearly superior, tracking closely with OPT, because the algorithm exploits the triangle inequality whenever possible to reduce backtracking. At low join sizes, it frequently identifies the optimal schedule. C-STA remains fairly competitive at low join sizes owing to the lack of backtracking for intra-cluster joins. However, C-STA does not attempt to optimize traffic on intra-cluster paths.

Figure 5 shows the computation time relative to *count* ∗. Because computation cost is linear in the number of tuples, *count* ∗ gives a lower bound. C-STA outperforms STA by over 10% for joins involving eight or more sites because intra-cluster join order minimizes the size of intermediate results. However, STA remains competitive even though it makes no attempt to optimize for query computation time. This is because the small intermediate results that lead to low network utilization also lead to low computation cost.

Figure 6 shows the utilization of paths with TCP throughput below the 3 Mbps threshold relative to STA. While C-STA is designed to reduce network utilization on narrow paths, it utilizes narrow paths more than STA. This arises because C-STA evaluates all sites within a cluster prior to sending data on narrow paths, whereas STA may

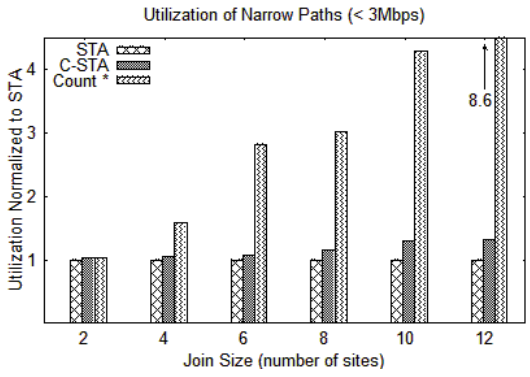

Figure 6: Utilization of narrow paths normalized to STA

send a small amount of data to another cluster and finish processing the rest of the cluster on its return (recall that most paths get traversed twice). Attribute aggregation contributes to the reduced performance, because C-STA aggregates all attributes in a cluster prior to sending results to the next cluster. It also receives less benefit from our reordering heuristic (Section 5.3), because clustering limits the ability to reorder intra-cluster joins. While C-STA is tunable via the clustering threshold, this artifact persists to high threshold values. Our future work will address these issues.

7 Discussion and Future Work

Latency-insensitive query processing requires new methods for query scheduling. Previous optimization methods focus on query response time, which degrade system throughput by overusing available resources [6]. Global-scale systems exacerbate the problem, because network paths are highly-heterogeneous and overusing a narrow path interferes with all subsequent queries that rely on that path. In a network-bound environment, maximum query throughput comes from balanced usage of all paths. We provide a balanced network utilization metric that captures this concept and two algorithms based on this metric.

To our surprise, the C-STA algorithm that balances network utilization and computation cost performs poorly on narrow paths. This is true even though it uses the same two-approximate solution as STA to optimize these paths. This arises because our assumption of no attribute aggregation is not accurate.

We are exploring the use of semi-joins to lessen the effect of attribute aggregation. By sending only join attributes between clusters, we can avoid sending non-join attributes over narrow paths twice. We believe that balancing network utilization and computation is important for scientific federations and tunable techniques to explore these trade-offs are needed. Our C-STA algorithm exposes these trade-offs and practical solutions to attribute aggregation are needed to make C-STA viable.

References

[1] CHEN, A., BRILL, D., TEMPLETON, M., AND YU, C. Distributed Query Processing in a Multiple Database System. *IEEE Journal on Selected Areas in Communications 7*, 3 (1989), 390–398.

[2] CHRISTOFIDES, N. Worst-case Analysis of a New Heuristic for the Traveling Salesman Problem. In *Symposium on New Directions and Recent Results in Algorithms and Complexity* (1976).

[3] CHUN, B., CULLER, D., ROSCOE, T., BAVIER, A., PETERSON, L., WAWRZONIAK, M., AND BOWMAN, M. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev. 33*, 3 (2003), 3–12.

[4] DESHPANDE, A., AND HELLERSTEIN, J. Decoupled Query Optimization for Federated Database Systems. In *ICDE* (2002).

[5] EVRENDILEK, C., DOGAC, A., NURAL, S., AND OZCAN, F. Multidatabase Query Optimization. *Distributed Parallel Databases 5*, 1 (1997), 77–114.

[6] GANGULY, S., HASAN, W., AND KRISHNAMURTHY, R. Query Optimization for Parallel Execution. In *SIGMOD* (1992).

[7] HOFFER, J., AND SEVERANCE, D. The Use of Cluster Analysis in Physical Database Design. In *VLDB* (1975).

[8] INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *MOBICOM* (2000).

[9] KOSSMANN, D., AND STOCKER, K. Iterative Dynamic Programming: A New Class of Query Optimization Algorithms. *ACM Transactions on Database Systems 25*, 1 (2000), 43–82.

[10] KRISHNAMACHARI, B., ESTRIN, D., AND WICKER, S. B. The Impact of Data Aggregation in Wireless Sensor Networks. In *ICDCSW* (2002).

[11] LEE, C., SHIH, C.-S., AND CHEN, Y.-H. Optimizing Large Join Queries Using A Graph-Based Approach. *IEEE Transactions on Knowledge and Data Engineering 13*, 2 (2001), 298–315.

[12] MALIK, T., BURNS, R., AND CHAUDHARY, A. Bypass Caching: Making Scientific Databases Good Network Citizens. In *ICDE* (2005).

[13] MALIK, T., SZALAY, A. S., BUDAVRI, A. S., AND THAKAR, A. R. SkyQuery: A Web Service Approach to Federate Databases. In *CIDR* (2003).

[14] MELIOU, A., CHU, D., HELLERSTEIN, J., GUESTRIN, C., AND HONG, W. Data Gathering Tours in Sensor Networks. In *IPSN* (2006).

[15] The Netperf Benchmark. http://www.netperf.org/netperf/.

[16] NG, T. E., AND ZHANG, H. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM* (2002).

[17] PADHYE, J., FIROIU, V., TOWSLEY, D., AND KRUSOE, J. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM* (1998).

[18] PRASAD, R., DOVROLIS, C., MURRAY, M., AND CLAFFY, K. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *IEEE Network 17*, 6 (2003), 27–35.

[19] SCHEUERMANN, P., AND CHONG, E. I. Distributed Join Processing Using Bipartite Graphs. In *ICDCS* (1995).

[20] SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, R. A., AND PRICE, T. G. Access Path Selection in a Relational Database Management System. In *SIGMOD* (1979).

[21] SHRIVASTAVA, N., BURAGOHAIN, C., AGRAWAL, D., AND SURI, S. Medians and Beyond: New Aggregation Techniques for Sensor Networks. In *SenSys* (2004).

[22] SZALAY, A., GRAY, J., THAKAR, A., KUNTZ, P., MALIK, T., RADDICK, J., STOUGHTON, C., AND VANDENBERG, J. The SDSS SkyServer - Public Access to the Sloan Digital Sky Server Data. In *SIGMOD* (2002).