Network-Aware Join Processing in Global-Scale Database Federations

Xiaodan Wang[1], Randal Burns[1], Andreas Terzis[1], Amol Deshpande[2]

[1] *Johns Hopkins University, USA*
{xwang,randal,terzis}@cs.jhu.edu

[2] *University of Maryland, USA*
amol@cs.umd.edu

***Abstract*— We introduce join scheduling algorithms that employ a balanced network utilization metric to optimize the use of all network paths in a global-scale database federation. This metric allows algorithms to exploit excess capacity in the network, while avoiding narrow, long-haul paths. We give a two-approximate, polynomial-time algorithm for serial (left-deep) join schedules. We also present extensions to this algorithm that explore parallel schedules, reduce resource usage, and define trade-offs between computation and network utilization. We evaluate these techniques within the SkyQuery federation of Astronomy databases using spatial-join queries submitted by SkyQuery's users. Experiments show that our algorithms realize near-optimal network utilization with minor computational overhead.**

I. Introduction

The evolution of database federations renders many of the goals of distributed query processing obsolete. Many previous distributed query processing works focus on minimizing query completion time. This includes parallel computation at multiple sites and reducing the volume of network traffic. 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 [1]. The goals of query scheduling should include minimizing computation at the member sites and, most importantly, the efficient use of network resources.

Federations are being built at a global scale and increasingly run data intensive applications, which examine tremendous amounts of data for each query. Data size and geography dictate that transmitting data takes large amounts of time and has a profound impact on system performance. The SkyQuery federation of Astronomy databases [2] typifies a data intensive, global-scale federation. It is not uncommon for join queries to yield intermediate results that are hundreds of megabytes in size [3]. Also, the physical distribution and connectivity of sites in SkyQuery vary tremendously with roughly thirty sites distributed across three continents. This federation will grow to over a hundred sites in the near future [4].

Our join scheduling algorithms optimize for the balanced utilization of all network paths. The cost model we use rewards schedules that utilize paths with excess capacity and produce small intermediate results. Join scheduling exploits network locality and avoids narrow, long-haul paths by making these paths more costly during optimization. For example, transferring data between sites that cross continental boundaries is highly undesirable, because paths that cross long physical distances exhibit higher propagation delay and lower throughput.

Contributions: We present a systematic study of the network issues involved in join processing in database federations. This includes identifying network structure, such as the throughput of paths and clusters of sites, and using this structure in query optimization. We also provide distributed join scheduling algorithms that find near-optimal schedules in heterogeneous networks under the assumption of perfect join selectivity. When compared with previous approaches, we simplify one aspect of query optimization (dealing with selectivities), which allows us to consider non-uniform and non-metric network costs and balance network utilization over all paths. Our algorithms have low, polynomial-time complexity and, thus, they scale to large federations with hundreds of sites.

We perform query optimization based on local information and aggregate statistics collected about the system prior to optimization. This allows for decentralized optimization decisions, achieving scale and incurring no communication overhead during optimization. Providing more timely knowledge during optimization about concurrent queries initiated from all sites in the network and the state of every network path would incur significant overhead and is not desirable. Deshpande et al. argued that federated database optimization techniques should minimize communication overhead when gathering cost information from the underlying data sources [5]. Bertsekas and Gallager showed that, in the context of shortest path routing algorithms, adapting to traffic conditions leads to rapid oscillatory behavior, which can degrade performance in dynamic environments [6]. One solution is to average traffic conditions over longer periods of time for each network path. We follow this approach in using aggregate statistics.

We present a two-approximate algorithm for serial (left-deep) join schedules that tracks closely with the network performance of an optimal algorithm. (A serial schedule is defined as a left-deep join order that visits each site serially). It increases computation minimally when compared with SkyQuery's current computation-optimizing scheduler. We also extend the two-approximate algorithm heuristically. Even

though network throughput is non-metric, we improve network performance by exploiting paths in which the triangle inequality holds. We cluster sites into regions of high-throughput, intra-cluster paths connected by low-throughput, inter-cluster paths and employ computation-optimizing schedules within the cluster and network-optimizing schedules among clusters. We also use a combination of semi-joins and bushy plans to reduce the negative effects of *attribute aggregation*: each site contributes new attributes to the intermediate result so that the result size grows throughout the schedule.

We evaluate our algorithms by running distributed queries over thirty sites on three continents, using both PlanetLab [7] and the SkyQuery federation. The workload comprises spatial joins across two to twelve sites and is derived from user queries extracted from SkyQuery's Web logs. Experimental results show that for large joins our techniques reduce network utilization by an order of magnitude when compared with SkyQuery's scheduler.

Our join scheduling techniques have applicability beyond Astronomy to many other scientific database federations and potentially to OLAP and decision support systems (DSS) in the future. SkyQuery pioneered data exploration through database federation. Since then, many other disciplines have followed suit, including Biology [8] and Geology [9]. Federation allows for queries against vast amounts of data that are stored and managed independently—data too large to be widely replicated or stored at a single site. As analysis and decision support move to larger data sets held across large geographies, we expect the same issues to emerge.

II. Related Work

Distributed query processing systems have changed significantly since early prototypes such as SDD-1 [10] and Distributed Ingres [11]. Kossmann [12] presents a detailed survey of both past and current query processing and optimization techniques that minimize computation and communication costs by exploiting, for instance, intra-query parallelism, caching, and data replication. Most commercial optimizers still rely on System R-style dynamic programming algorithms, which enumerate and then prune a set of query plans [13]. These algorithms guarantee optimal solutions, but have exponential-time complexity [5][14], which makes them unsuitable for query optimization in a large-scale database federation.

Distributed query optimization remains a hard problem [12], but polynomial-time algorithms exist that provide good empirical results and scale to large joins. Iterative Dynamic Programming [13] divides complex queries into smaller subproblems so that applying dynamic programming is computationally tractable. However, adapting existing algorithms to account for network heterogeneity is a non-trivial problem that we plan to investigate in future work.

Many works that address the communication costs of distributed queries assume network uniformity [15][16]. They minimize the size of intermediate results, which in turn reduces computation and network costs on uniform networks. While suitable for local area networks, query optimization over wide-area networks invalidates the network uniformity assumption.

The joint optimization of computation and network costs can be used to minimize query response time. Mermaid [17] models both transmission cost and execution cost. Evrendilek et al. [18] study the problem in the presence of data replication and account for the execution time of queries at runtime. However, minimizing the response time of joins through parallel execution sacrifices job throughput and leads to unbalanced resource allocation [1].

Evaluating distributed queries using semi-joins can provide substantial network savings [15][19][20]. Semi-joins ship only attributes that are necessary for evaluating a join to another site in order to eliminate tuples that fail to satisfy the join predicate. In many settings, semi-joins are not attractive because computational overhead outweighs network savings on local area networks [21]. For network-bound queries on wide-area networks, semi-joins become more attractive because communication costs dominate performance. Our application of semi-joins differs in that we use them to limit attribute aggregation, rather than reducing the cardinality of intermediate results.

Data-centric routing in wireless sensor networks shares some techniques with our work. Several works perform in-network aggregation of data from multiple sensors to a single base station [22][23][24]. They organize sensors into a spanning tree rooted at the base station and aggregate data along the tree in order to conserve power by minimizing network usage. We explore more complex network structure, variable-size intermediate results, and employ dynamic programming on top of spanning tree solutions.

Also related is the work of Meliou et al. [25], which studies the NP-hard problem of finding an optimal tour for gathering data from a subset of sensors. They employ a similar metric to capture network utilization, but solve a different problem with different techniques: optimizing power consumption using dynamic programming alone.

III. Query Scheduling in SkyQuery

The SkyQuery [2] federation of Astronomy databases makes data available to the public through Web services and Web forms. Users submit read-only queries to 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.

Cross-match [2], the principal query in SkyQuery, joins observations of the same astronomical object from several databases by correlating their location in space. The `from` clause includes an unordered list of databases to visit. Cross-match also extends SQL by adding two clauses. The `region` clause specifies an area for conducting the search, and the `xmatch` clause defines an error bound, in standard deviations, for a probabilistic spatial join (Figure 1). Because the measured positions of the same object differ among databases,

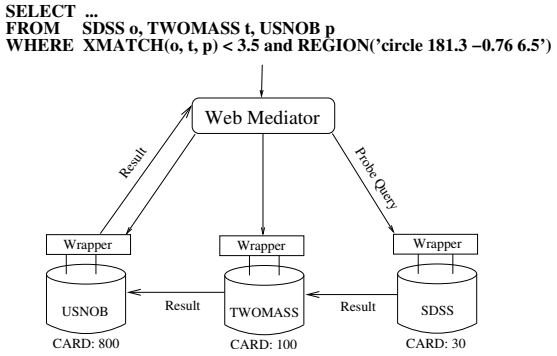


Fig. 1. Spatial join in SkyQuery.

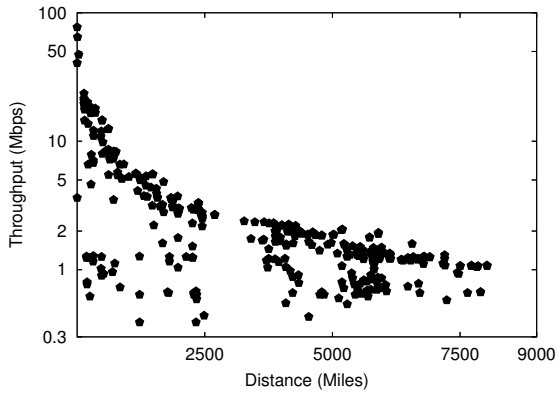


Fig. 2. Distance vs. TCP throughput.

`xmatch` specifies a threshold at which different observations are joined as a common object.

SkyQuery employs a count $*$ [2] approach to produce join schedules for cross-match queries. The mediator issues probe queries to all sites participating in the join to determine site cardinality, defined as the number of rows that satisfy the `region` clause. 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 that joins sites by ascending cardinality.[1]

The count $*$ approach is effective at the early elimination of tuples that do not participate in the join at subsequent sites. This approach minimizes computation time and I/O for databases on uniform networks under the assumptions of perfect join selectivity and linear I/O and processing costs in the number of tuples. Perfect join selectivity means that every tuple at a lower cardinality site produces a match with tuples at a higher cardinality site. Instrumenting SkyQuery shows that these assumptions hold. However, the insensitivity of count $*$ to geography results in poor schedules. By scheduling using cardinality alone, count $*$ may send large amounts of data back and forth across narrow, intercontinental network paths.

IV. Minimizing Network Utilization

Our goal in query scheduling is to reduce query delay and minimize the utilization of network resources by large-scale federations such as SkyQuery, which processes up to a million queries each month. We do so by biasing join schedules toward high capacity network paths. We define a path's capacity as the throughput achieved by a single TCP connection, also known as the path's available bandwidth. A TCP-specific performance metric is appropriate because SkyQuery uses TCP, and TCP carries nearly 90% of the Internet traffic [27].

Given a join schedule over n sites, we define the *balanced network utilization* metric as:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{VOL_{ij} + VOL_{ji}}{TP_{ij}} \quad (1)$$

[1]Recent techniques [26] make it possible to accurately estimate the number of rows contributed by each site in SkyQuery, which will obviate the need for probe queries.

VOL_{ij} denotes the volume of data transmitted from site i to site j. TP_{ij} is the average path throughput between sites i and j. We assume symmetric paths to simplify our analysis; we observed only minor asymmetry on the paths that were measured. Inverse TCP throughput measures the time cost of transmitting data (per byte) on that path. For example, 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. Notice that the proposed metric is expressed in units of time. In other words, this metric captures the total cost, in terms of time, of using the selected network paths to send data for the join query.

By minimizing this metric, we minimize response time for serial schedules by exploiting network heterogeneity via the selection of high throughput paths. Moreover, for all schedules, minimizing this metric reduces the total time in which network's paths are used to carry data associated with join queries, thereby reducing contention with existing network traffic. Similar metrics were used by Chu et al. [28] for application-level multicast protocols and more recently by Pietzuch et al. [29] in the context of operator placement in distributed stream-processing. In both works, the cost of utilizing a network path is defined as the product of path latency and data size. Our metric replaces latency with inverse of TCP throughput for two reasons: throughput is inversely proportional to latency [30] and our workload performs bulk data transfer for which throughput is the natural measure. A similar formulation is used in the context of optimal routing for computer networks by Bertsekas and Gallager [6].

To calculate the metric in Equation 1, we estimate the pair-wise throughput for all SkyQuery sites. We approximate throughput by measuring TCP throughput at PlanetLab [7] sites that are close (in the network) to the corresponding SkyQuery sites. PlanetLab proxies are used because we do not have access to all SkyQuery sites. Figure 2 plots geographical distance against TCP throughput for 400 node pairs. Measurements are obtained over a one-hour period using the *netperf* benchmark [31] in which we take the average of three measurements for each pair of nodes. The result shows a dominant trend in which TCP throughput is inversely proportional to distance. This is consistent with the model of

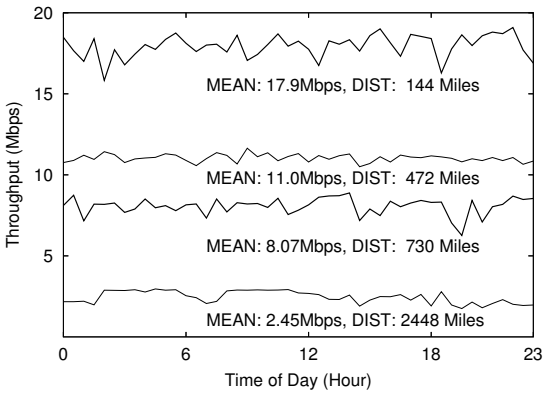


Fig. 3. Available throughput variation.

Padhye et al. [30] for bulk transfer TCP flows, which shows that throughput is inversely proportional to the round-trip time (RTT) of a given path.

Despite significant network heterogeneity, 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. However, some physically close sites exhibit low pair-wise throughput. The root cause of this anomaly is that RTT in the Internet is non-metric and does not always correlate with physical distance [32]. This prevents us from relying on the metric property for scheduling. However, we do identify paths over which the triangle inequality holds and take advantage of metric optimizations when available.

To obtain available throughput between each pair of sites, we measure average throughput for large file transfers. Because measurements are intrusive due to the amount of traffic they generate, we sample paths infrequently. Paths can also be measured by taking frequent, short-duration samples. However, the latter method exhibits high variability due to cross-traffic [27] and is an unreliable performance predictor for the transfer of large query results. Figure 3 indicates that average available throughput on the paths we measured remains fairly stable over time. Even though these paths are among the most volatile over a 24-hour period, throughput measurements never deviate more than 30% from their mean value. Once throughput values between all pairs of sites are collected, they are stored at the mediator for join scheduling. The relatively low variability in throughput means that new measurements are collected only a few times per day. We also exploit information from actual TCP transfers at each site to update the available throughput estimates.

V. Scheduling Algorithms

We describe two base algorithms that optimize for balanced network utilization. The spanning tree approximation (STA) adapts the two-approximate solution to the traveling salesman problem (TSP) to query scheduling. By exploiting near perfect join selectivity in SkyQuery, STA achieves a two times bound on the optimal serial schedule (SerOpt), which is determined by an exhaustive search of all feasible serial schedules that visit each join site exactly once and end at the mediator. We also present a clustering algorithm (C-STA) that extends STA by clustering sites based on pair-wise path capacity and then uses the count $*$ approach within the clusters and STA among the clusters. This optimizes computation and I/O in well-connected network regions and achieves balanced network utilization over narrow paths.

In addition, we provide two extensions to the base algorithms that further reduce utilization in exchange for computational overhead. By introducing semi-joins (STA-SJ and C-STA-SJ), we ensure that the base algorithms avoid penalties from attribute aggregation. This is accomplished by only sending join attributes away from the root of the spanning tree. Finally, we present polynomial-time dynamic programming algorithms (STA-BP and C-STA-BP) that reduce utilization further by exploring bushy plans. By restricting bushy plans to the edges of the MST, we avoid a combinatorial explosion in the problem space.

We exploit assumptions about join selectivities and attribute aggregation to derive polynomial-time approximation algorithms for join scheduling. Rather than estimate the selectivity of arbitrary n-way joins, we take a conservative approach and assume perfect join selectivity in which the join of three relations with r_m, r_n, and r_o tuples produces a result with $min(r_m, r_n, r_o)$ tuples. Astronomy queries satisfy perfect join selectivity 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. We assume no attribute aggregation for our base algorithms, allowing us to map the join scheduling problem to TSP. However, attribute aggregation is factored back into query optimization in the semi-join and bushy plan extensions.

A. Spanning Tree Approximation (STA)

We achieve a two-approximation of SerOpt by adapting a solution to metric TSP [33], which uses a minimum spanning tree (MST). Even though TCP throughput is non-metric, the solution applies because (unlike TSP) a join schedule can visit each site more than once.

Our algorithm takes the minimum cardinality site and convolves its data among all sites, following paths in the MST. More formally, provided a mediator site S_1 and join sites S_2 through S_n, let r_i be the number of rows that fall inside the query region for a site S_i. (The mediator site initiates the query and receives the results). Let r_{min} be $min(r_2, ..., r_n)$ 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 of $S_1, \ldots, S_n$. Pick the mediator, S_1, as root of the spanning tree and recursively enumerate a breadth-first order that visits all nodes in each of the sub-trees rooted at its children. These serve as join schedules for each sub-tree. Finally, combine the join schedules so that the final schedule starts at S_{min}, finishes at S_1, and visits each site at least once.

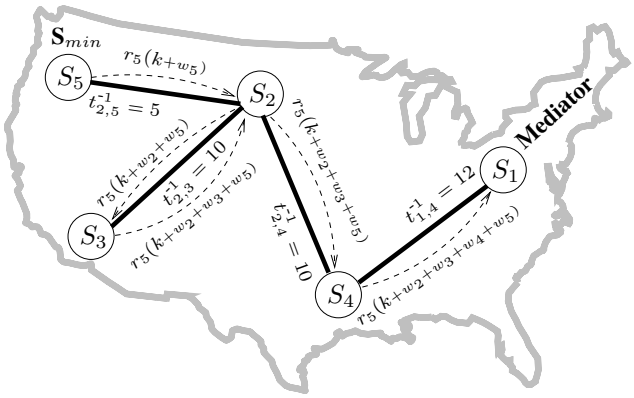


Fig. 4. Sample join schedule.

	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**

TABLE I
CLUSTERING TCP THROUGHPUT (MBPS).

The network cost of the optimal serial schedule, $cost(\mathsf{SerOpt})$, is at least as large as the sum of edge cost of the minimum spanning tree, $cost(\mathrm{MST})$, because each site is visited once. STA produces schedules that traverse each edge on the MST at most twice: once to visit all descendant nodes and once to return to the parent. The edges connecting S_{min} to S_1 are traversed only once. Under the assumptions of no attribute aggregation and perfect join selectivity, it follows that

Theorem 1: $cost(\mathsf{STA}) \leq 2 \cdot cost(\mathsf{SerOpt})$.

Although throughput is non-metric, the triangle inequality holds frequently. Thus, STA avoids backtracking when the direct path to the next node in the schedule costs less than the path on the MST. By exploiting metric paths and bypassing previously visited nodes in the join schedule, we further reduce network cost by up to 30%. However, this does not improve the algorithmic bound.

Figure 4 shows a serial schedule that employs the edges of a MST. The inverse of TCP throughput for the path connecting S_i and S_j is given by $t_{i,j}^{-1}$. For a site S_i, let w_i denote the total byte-width of all non-join attributes that S_i contributes to the join result. k is the byte-width of join attributes. The join schedule begins at S_5 and proceeds to S_2 then S_3. It backtracks to S_2 before continuing to S_4 and S_1. The schedule aggregates attributes with byte-width w_i from each site S_i on its path.

B. Clustered STA (C-STA)

PlanetLab measurements show that sites located in close geographic proximity form well-connected components. Thus, we can cluster sites based on TCP throughput so that intra-cluster paths have high, relatively uniform throughput when compared with inter-cluster paths.

We employ the bond energy algorithm (BEA) [34] for clustering SkyQuery sites. BEA operates on object-object data arrays and reorders the rows and columns in order to maximize the similarity between the values of neighboring entries. BEA runs in $O(n^2)$ for n sites and uses $\mathrm{O}(n^2)$ space. We populate the data array with pairwise TCP throughput and then reorder the array using BEA to maximize the similarity among neighboring entries. Finally, we extract groups of related nodes from the array using a threshold. Table I illustrates the result of applying BEA on the TCP throughput data array for a subset of nodes: ten 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 thirty PlanetLab proxy nodes.

The clustered spanning tree approximation (C-STA) hierarchically combines count $*$ with 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 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. 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 STA. Tuning the threshold offers a family of algorithms that explore trade-offs between computation and network utilization. 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. We revisit the issue of tuning this threshold in Section VII.

C. Semi-Join Extension (STA-SJ/C-STA-SJ)

Semi-joins can reduce the negative effects of attribute aggregation. By sending only join attributes down the tree, tuples that fail to satisfy the join predicate are eliminated without incurring the cost of sending a large number of non-join attributes away from the root. Semi-joins have been used in this fashion in other distributed query processing systems [15][19][20].

Sending join attributes down the tree ensures that the number of tuples that traverse edges on the tree remains unchanged from STA, but the corresponding edge in STA may be more costly. On the path between sites S_2 and S_3 in

Figure 4, the amount of traffic sent down the tree and back are $r_5(k+w_2+w_5)$ and $r_5(k+w_2+w_3+w_5)$ respectively in STA. The corresponding traffic is reduced to $r_5 k$ and $r_5(k+w_3)$ in STA-SJ, because non-join attributes from S_2 and S_5 do not traverse the path. In fact, the network cost on each path is no worse than that used in our analysis of STA. Thus, STA-SJ is also a two-approximate algorithm.

Unlike STA, STA-SJ cannot exploit metric regions with respect to TCP throughput in order to avoid backtracking on the MST. Thus, $cost(\text{STA-SJ})$ and $cost(\text{STA})$ are not directly comparable. Either may find the better solution.

Although the semi-join extension reduces the impact of attribute aggregation, it incurs processing overhead. In STA, a `select-project` query is first executed at the minimum cardinality site on the area specified by the `region` clause. The result is then sent, via paths along the MST, to the next site, which executes a join on the indexed, spatial attributes [2]. This join is repeated at the remaining sites, which incurs $n-1$ join operations for n sites. However, in STA-SJ, each site employs semi-joins to collect join results from its children and then merge the intermediate results. Thus, a non-leaf site with c children incurs $c-1$ additional joins compared with STA. In order to join S_2 with its children in Figure 4, S_2 first joins tuples from S_5. S_3 then performs a semi-join using join attributes from S_5 before the two intermediate results merge at S_2.

D. Bushy Plans (STA-BP/C-STA-BP)

We describe a polynomial-time algorithm that generates bushy plans, which further reduce network utilization. Bushy plans evaluate sub-trees of the MST in parallel without requiring inputs from the minimum cardinality site. We use dynamic programming to choose between parallel and semi-join evaluations at each level of the MST, which produces the minimum cost join schedule using only edges on the MST. Restricting the solution to the MST conforms to our goal of exploiting network paths with high TCP throughput, produces good solutions, and keeps the problem space manageable.

Compared to serial plans, bushy plans avoid backtracking on the MST. A bushy plan chooses the minimum cardinality site within the sub-tree and convolves that site around the sub-tree. The join result of that sub-tree are merged with results from outside the sub-tree at the sub-tree's parent in the same way that semi-join results are merged. While the minimum cardinality site in the sub-tree contains more rows than the global minimum, it may produce a lower cost schedule than a serial plan.

We use the following notation in our formulation. Let the spanning tree be rooted at S_1. The children of node $S_\mathbf{i}$ are $S_{\mathbf{i}1}, S_{\mathbf{i}2}$, etc. (Bold $\mathbf{i}$ indicates a vector quantity, *e.g.* $\mathbf{i}=121$). The cost or inverse of TCP throughput of an edge between sites $S_\mathbf{i}$ and $S_{\mathbf{i}1}$ is $t^{-1}_{\mathbf{i},\mathbf{i}1}$. For each sub-tree rooted at $S_\mathbf{i}$, define $r_\mathbf{i}$ as number of rows at the minimum cardinality site local to the sub-tree, $w_\mathbf{i}$ as the total byte-width of non-join attributes that sites in the sub-tree contribute to the join result, and k as the byte-width of join attributes.

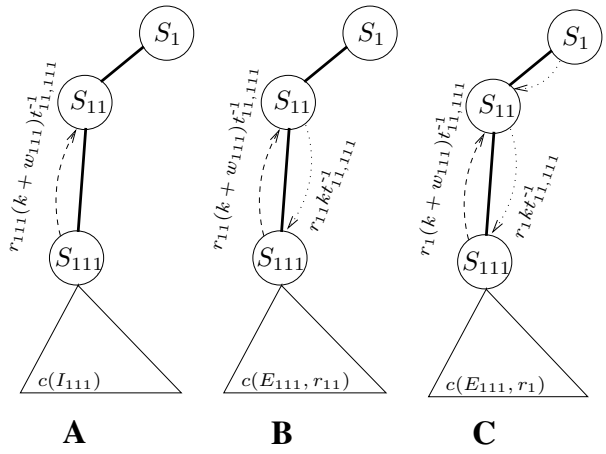


Fig. 5. Join strategies for sub-tree S_{111}.

Figure 5 illustrates three strategies for joining the sub-tree rooted at S_{111} to its parent. Strategy **A** joins the sub-tree in parallel in which all sites in a sub-tree are joined independently using data local to the sub-tree. The root S_{111} of the sub-tree returns to S_{11} the join result, which is bounded in size by r_{111} rows times the byte-width of all attributes from the sub-tree or $k+w_{111}$. In strategies **B** and **C**, a semi-join is used to join the sub-tree rooted at S_{111} involving the minimum cardinality site from sub-trees S_{11} and S_1 respectively. The amount of data that traverse the path between S_{11} and S_{111} in **B** is $r_{11}(2k+w_{111})$. ($2k$ because join attributes traverse the edge twice).

The dynamic program computes for each sub-tree (1) the cost of processing the sub-tree in parallel and (2) the cost of waiting for the parent to initiate a semi-join containing fewer rows than the local minimum cardinality site. Let $c(I_\mathbf{i})$ denote the minimum cost schedule for joining a sub-tree rooted at $S_\mathbf{i}$ in parallel, and $c(E_\mathbf{i}, X)$ denote the cost of joining the sub-tree provided that the parent of $S_\mathbf{i}$ initiates a semi-join containing X rows. Multiple semi-join strategies are evaluated, because a semi-join may be initiated from different levels in the tree: the parent, grandparent, etc.

Our dynamic programming algorithm for balanced network utilization computes the minimum cost of joining each sub-tree for each join strategy. Let $S_\mathbf{i}$ be a join site with c children $S_{\mathbf{i}1}, \ldots, S_{\mathbf{i}c}$ and $\boldsymbol{P}_\mathbf{i}$ be the set of sites on the path from $S_\mathbf{i}$ to S_1 such that $S_\mathbf{x}$ is an ancestor of $S_\mathbf{i}$ only if $S_\mathbf{x} \in \boldsymbol{P}_\mathbf{i}$. The following recurrence defines the cost of the semi-join and parallel join strategies for a sub-tree rooted at $S_\mathbf{i}$:

$$c(I_\mathbf{i}) = \sum_{j=1}^{c} min(c(I_{\mathbf{i}j}) + r_{\mathbf{i}j}(k+w_{\mathbf{i}j})t^{-1}_{\mathbf{i},\mathbf{i}j}, \\ c(E_{\mathbf{i}j}, r_\mathbf{i}) + r_\mathbf{i}(2k+w_{\mathbf{i}j})t^{-1}_{\mathbf{i},\mathbf{i}j})$$

$$c(E_\mathbf{i}, r_\mathbf{x}) = \sum_{j=1}^{c} min(c(I_{\mathbf{i}j}) + r_{\mathbf{i}j}(k+w_{\mathbf{i}j})t^{-1}_{\mathbf{i},\mathbf{i}j}, \\ c(E_{\mathbf{i}j}, r_\mathbf{x}) + r_\mathbf{x}(2k+w_{\mathbf{i}j})t^{-1}_{\mathbf{i},\mathbf{i}j}) \\ \forall S_\mathbf{x} \in \boldsymbol{P}_\mathbf{i}$$

This formulation has optimal substructure, namely, the minimum cost solution for a sub-tree rooted at $S_\mathbf{i}$ always

chooses the minimum cost join strategy for each of its children. We omit the proof due to space limitations.

The solution to $c(I_1)$ gives the total cost of the optimal schedule over all join sites and can be computed in polynomial time. The perfect join selectivity assumption helps bound the number of subproblems, because only the number of rows from the minimum cardinality site of each sub-tree is used to compute the semi-join cost (in Figure 5, only r_{11} is considered by strategy **B**). For a sub-tree rooted at $S_\mathbf{i}$, the algorithm must solve for $c(I_\mathbf{i})$ and $c(E_\mathbf{i}, r_\mathbf{x})$ $\forall S_\mathbf{x} \in \boldsymbol{P}_\mathbf{i}$, in which $\boldsymbol{P}_\mathbf{i}$ is the set of sites on the path from $S_\mathbf{i}$ to S_1. In joins involving n sites, the algorithm runs in $O(n^2)$ time, because the size of $\boldsymbol{P}_\mathbf{i}$ is bounded by $S_\mathbf{i}$'s depth in the spanning tree. $O(n^2)$ space is required to store the minimum cost solution for the various join strategies of each sub-tree.

VI. Experiments

We implement all algorithms in the SkyQuery federation and instrument their performance using a workload of cross-match (SkyQuery spatial join) queries derived from user queries submitted to the system. We also compare the balanced network utilization of our algorithm against the optimal serial schedule (SerOpt). We do not compare against the optimal bushy plan, because finding the plan is computationally infeasible for even a handful of sites [35]. In addition, we compare with SkyQuery's count $*$, which provides a lower bound on the computational performance of a schedule under our performance assumptions: perfect join selectivity and linear computation costs with the number of tuples to be joined.

Running experiments on SkyQuery itself adds some complexity. We do not have full access to all sites in the federation and cannot extract detailed performance information from all sites. Thus, we measure computation costs on replicas of the member databases stored in the computing cluster at the Sloan Digital Sky Survey [36] and network costs on the PlanetLab proxy sites. Member databases range from hundreds of gigabytes to several terabytes in size. Queries are executed on Windows Server 2003 workstations equipped with 2.5GB of memory.

A. Workload and Baseline Results

Figure 6 shows the network utilization performance of all algorithms on a ten-thousand query trace from the SkyQuery Web logs. Our algorithms bests count $*$ by up to 50%. While these gains are substantial, this workload does not exercise varied or large network topologies; 98% of queries join two or three sites, and a vast majority of queries join among the three largest sites in the federation: `twomass`, `sdss`, and `sdssdr2`. This workload exhibits homogeneous access patterns because of performance issues with the current query processing techniques and the recent growth of the federation, which have prevented users from joining more than five sites with any frequency. Thus, the ten thousand queries represent a much less diverse workload than we would like.

To explore network heterogeneity, we require a workload with queries that join a large number of geographically diverse sites. We accomplish this by taking thirty representative queries from the trace and using these as templates to produce queries with varying join sizes and spanning tree topologies. To introduce diverse network topologies, we substitute sites in the original query with randomly selected sites. We also increase the join size by adding randomly selected sites to the original queries, which allows us to compare performance in a larger federation by scaling to higher join sizes.

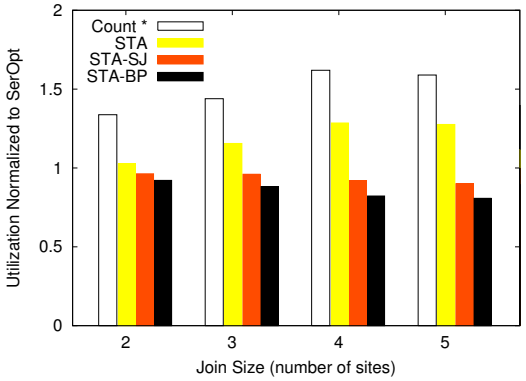


Fig. 6. Network utilization for a 10k query trace.

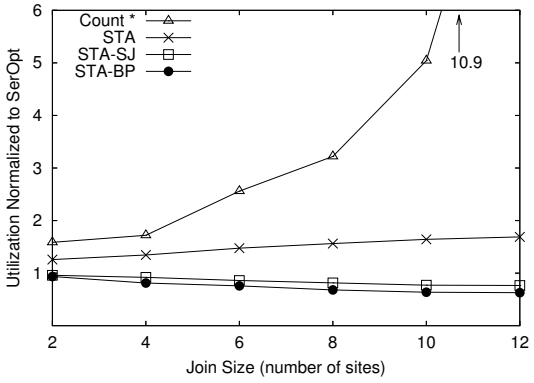


Fig. 7. Network utilization.

Although augmenting the workload is less than ideal, our intent is to conduct a forward looking performance and scalability study that is faithful to the geography and science of SkyQuery. At present, a select group of sites dominate the SkyQuery federation. However, the utility of federated scientific databases lies in the exploration of many disparate data sources. In the context of SkyQuery, Jim Gray observed that the number of scientific discoveries increases polynomially in the number of different data sources [37]. New sites emerge every year and recent standards [38] should increase the federation's usability.

B. Results

Figure 7 shows the network utilization for the STA family of algorithms in queries involving two to twelve sites. The results at each join size represent an average of 100 queries and are normalized to the performance of SerOpt.

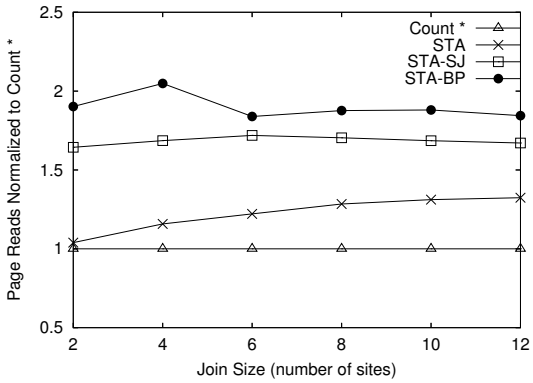


Fig. 8. Query computation cost.

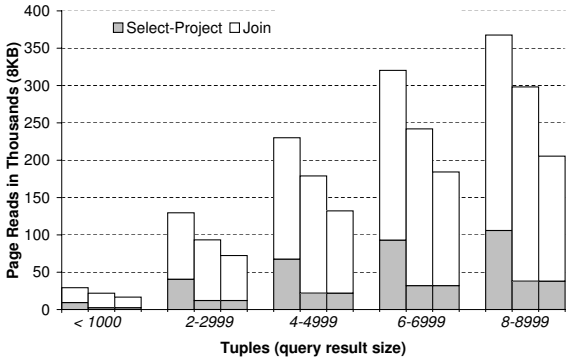


Fig. 9. Decomposition of computation cost for STA-BP (left bar), STA-SJ (center bar), and count $*$ (right bar).

STA tracks closely with the network performance of SerOpt. Like SerOpt, STA generates a serial schedule. However, it does not include the cost of attribute aggregation in searching for a plan. Despite this, STA never exceeds SerOpt by more than 1.8 times. This validates our two main assumptions regarding SkyQuery: perfect join selectivity and the number of rows dominate data transfers. STA frequently identifies the same schedule as SerOpt because it exploits the triangle inequality whenever possible to avoid backtracking. In contrast, the network utilization of count $*$ scales poorly with join size. It is over six times worse than STA in joins involving twelve sites because it does not consider network heterogeneity. We note that count $*$ produces the smallest volume of network traffic, but it tends to over-utilize narrow paths and does not exploit network locality. Our results show that it is not sufficient to simply minimize the size of intermediate results.

Semi-joins and bushy plans improve network usage, even beyond SerOpt. They perform roughly 15 (STA-SJ) and 20 (STA-BP) times better than count $*$. Given the performance of STA-SJ, most of the benefit comes from sending only join attributes away from the root of the MST. Bushy plans improve network utilization further, but only a little. This is because parallel schedules avoid semi-joins only if the cost of propagating the join attributes down the tree outweighs the benefits. The cost of sending join attributes is only a fraction of total utilization, which limits the opportunity for parallel evaluation of sub-trees.

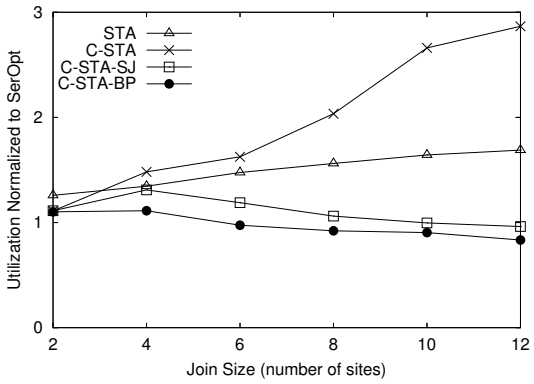
The benefits of semi-joins and bushy plans come at the expense of increased computation (Figure 8). These plans use 1.5 to 2 times as much computation as does count $*$. In contrast, STA uses about 30% more computation. STA is competitive even though it makes no attempt to optimize for query computation time (*e.g.* sites are not joined in increasing cardinality order). This is because the small intermediate results that lead to low network utilization also lead to low computation costs.

STA-SJ incurs higher computational overhead due to the additional joins described in Section V-C. Bushy plans incur even more computational overhead. Figure 9 reveals the distribution of this overhead by decomposing the cost of query evaluation for joins involving three sites. Unlike STA-SJ in which a `select-project` query is executed only once at the minimal cardinality site, bushy schedules may process sites in parallel and incur overhead for additional `select-project` queries. Moreover, the join overhead from STA-BP is comparable to STA-SJ despite relying less on semi-joins. This is because bushy schedules do not always convolve rows from the minimum cardinality site such that each branch may generate up to twice as many tuples as the minimum cardinality site. The resulting joins operate on larger data sets and are less-efficient.

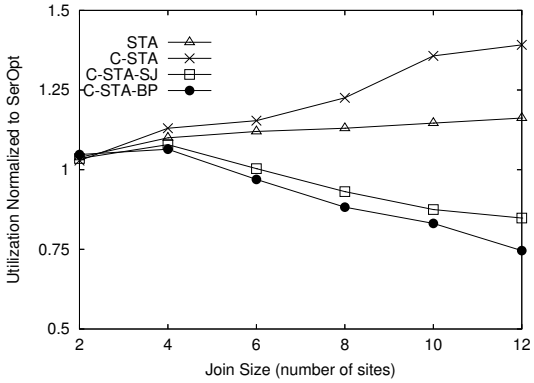
Clustering algorithms exhibit even more sensitivity to attribute aggregation. Figure 10(a) shows network utilization for C-STA and its variants normalized to SerOpt. We include the performance of STA for comparison. C-STA exhibits substantially worse network utilization than does STA. We note that C-STA does not attempt to optimize the intra-cluster paths, which accounts for some of the difference. Surprisingly, C-STA has similar overhead on the narrow paths ($<$3 Mbps), which it does attempt to optimize. Figure 10(b) shows the utilization on inter-cluster, narrow paths only. Attribute aggregation contributes to the reduced performance on narrow paths because C-STA evaluates all sites and aggregates all attributes in a cluster prior to sending results across narrow, inter-cluster paths. However, STA may send a small amount of data to another cluster and finish processing the rest of the cluster on its return (many paths are traversed twice).

As with STA, the use of semi-joins addresses attribute aggregation in C-STA. It negates the penalty of visiting all sites within a cluster prior to using a narrow path. The size of the results sent down the MST is the same regardless of how many sites are visited and all attributes must go up to the root from a cluster in all plans. Network utilization for C-STA-SJ and C-STA-BP follows that of the non-clustering algorithms.

Using semi-joins and bushy plans, clustering achieves comparable network performance at lower computational overhead. Figure 11 compares performance of all algorithms for twelve sites. Clustering accomplishes its goal of reducing computation through count $*$ scheduling within clusters. However,



(a) All paths



(b) Narrow paths (<3 Mbps)

Fig. 10. Network utilization for clustering algorithms.

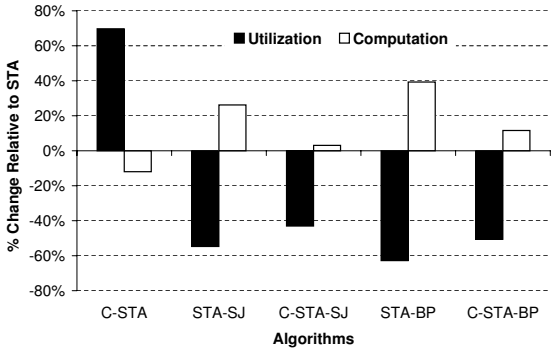


Fig. 11. All algorithms compared (12 sites).

the benefit is modest and comes at a modest increase in network utilization.

Looking at all the algorithms together reveals structure about the relative benefits of each technique. We normalize all results to STA because of its balanced performance. Other algorithms present computation versus network utilization tradeoffs from this point. By itself, clustering is not an attractive technique, increasing network utilization without improving computation commensurately. As discussed, attribute aggregation is the culprit. Semi-joins address attribute aggregation at a small increase in computation. Semi-joins alone realize most of the benefits, while bushy plans further these gains. However, bushy plans do not dominate semi-joins. Adding clustering minimizes the computation costs. Combining bushy plans and/or clustering with semi-joins explores the computation versus network utilization parameter space.

VII. Discussion

Our implementation led to observations on the nature of join scheduling. We also address the application of these techniques to other workloads and systems.

The explicit clustering of sites did not have as profound an effect on performance as we initially thought, which we attribute to "natural" clusters of sites that exist in SkyQuery's deployment. Thus, tuning the clustering threshold does not lead to a continuous family of algorithms in practice. Clustering does reduce computational overhead, but the performance trade-offs are small. Sites close together have near-uniform throughput and tend to get visited together by all algorithms: explicitly by clustering algorithms and by virtue of the high-throughput paths for the other algorithms. We expect clustering to be of more import on larger federations with queries that join more sites.

Our treatment restricts join processing to edges on the MST. However, our techniques apply equally well to any spanning tree. We have not considered the implications of optimizing on arbitrary spanning trees or introducing edges outside of the MST. These are interesting areas that we plan to explore in the future. Specifically, shallow trees may offer better network performance at the expense of increased computation.

Bushy plans and semi-joins lead to an unfair distribution of computation costs. Strictly serial schedules assign computation to a site in proportion to the data it contributes. Depending on the topology of the federation, bushy plans and semi-joins assign a disproportionate amount of work to sites with a large number of children. Network topology and an unequal distribution of incoming queries may lead to load skew in the federation.

Join scheduling based on balanced network utilization holds promise for applications outside of scientific databases. With an increasing need for real-time processing over data generated all over the globe, OLAP and DSS applications are starting to employ federations rather than summary databases and data warehouses [39][40]. The challenges faced in such federations will be quite similar to SkyQuery with a need to balance the network utilization across heterogeneous paths. Furthermore, the perfect join selectivity assumption used in our join algorithms might either hold or be a reasonably good approximation on many OLAP and DSS queries, *e.g.* joins on foreign keys or attributes with uniqueness constraints. A core premise of this research is that capturing network heterogeneity is more important than the inaccuracies that arise from selectivity assumptions. Our experiments bear this out for SkyQuery.

The balanced network utilization metric captures path heterogeneity in large-scale federations and, thus, can be used for

all widely-distributed, data intensive applications. At present, the quality of join schedules depends on the accuracy of selectivity estimates and join probabilities. Applications in different domains may require new algorithms that better capture their specific properties. However, scheduling based on balanced network utilization will ensure that optimization results in schedules that use network resources efficiently.

REFERENCES

[1] S. Ganguly, W. Hasan, and R. Krishnamurthy, "Query Optimization for Parallel Execution," in *SIGMOD*, 1992.

[2] T. Malik, A. S. Szalay, A. S. Budavri, and A. R. Thakar, "SkyQuery: A Web Service Approach to Federate Databases," in *CIDR*, 2003.

[3] T. Malik, R. Burns, and A. Chaudhary, "Bypass Caching: Making Scientific Databases Good Network Citizens," in *ICDE*, 2005.

[4] A. Szalay, J. Gray, A. Thakar, P. Kuntz, T. Malik, J. Raddick, C. Stoughton, and J. Vandenberg, "The SDSS SkyServer - Public Access to the Sloan Digital Sky Server Data," in *SIGMOD*, 2002.

[5] A. Deshpande and J. Hellerstein, "Decoupled Query Optimization for Federated Database Systems," in *ICDE*, 2002.

[6] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1992.

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, 2003.

[8] The Human Brain Project Database. [Online]. Available: http://ycmi-hbp.med.yale.edu/hbpdb/

[9] Chronos. [Online]. Available: http://www.chronos.org/index.html

[10] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. James B. Rothnie, "Query Processing in a System for Distributed Databases (SDD-1)," *ACM Trans. Database Syst.*, vol. 6, no. 4, pp. 602–625, 1981.

[11] M. Stonebraker, "The Design and Implementation of Distributed INGRES," in *The INGRES Papers: Anatomy of a Relational Database System*. Addison-Wesley Longman Publishing Co., Inc., 1986, pp. 187–196.

[12] D. Kossmann, "The State of the Art in Distributed Query Processing," *ACM Comput. Surv.*, vol. 32, no. 4, pp. 422–469, 2000.

[13] D. Kossmann and K. Stocker, "Iterative Dynamic Programming: A New Class of Query Optimization Algorithms," *ACM Trans. on Database Systems*, vol. 25, no. 1, pp. 43–82, 2000.

[14] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access Path Selection in a Relational Database Management System," in *SIGMOD*, 1979.

[15] A. L. P. Chen and V. O. K. Li, "Optimizing Star Queries in a Distributed Database System," in *VLDB*, 1984.

[16] P. Scheuermann and E. I. Chong, "Distributed Join Processing Using Bipartite Graphs," in *ICDCS*, 1995.

[17] A. Chen, D. Brill, M. Templeton, and C. Yu, "Distributed Query Processing in a Multiple Database System," *IEEE J. Select. Areas Commun.*, vol. 7, no. 3, pp. 390–398, 1989.

[18] C. Evrendilek, A. Dogac, S. Nural, and F. Ozcan, "Multidatabase Query Optimization," *Distributed Parallel Databases*, vol. 5, no. 1, pp. 77–114, 1997.

[19] D.-M. Chiu, P. A. Bernstein, and Y.-C. Ho, "Optimizing Chain Queries in a Distributed Database System," *SIAM J. Comput.*, vol. 13, no. 1, pp. 116–134, 1984.

[20] Y. Kambayashi, M. Yoshikawa, and S. Yajima, "Query Processing for Distributed Databases using Generalized Semi-Joins," in *SIGMOD*, 1982.

[21] H. Lu and M. J. Carey, "Some Experimental Results on Distributed Join Algorithms in a Local Network," in *VLDB*, 1985.

[22] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *MOBICOM*, 2000.

[23] B. Krishnamachari, D. Estrin, and S. B. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," in *ICDCSW*, 2002.

[24] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and Beyond: New Aggregation Techniques for Sensor Networks," in *SenSys*, 2004.

[25] A. Meliou, D. Chu, J. Hellerstein, C. Guestrin, and W. Hong, "Data Gathering Tours in Sensor Networks," in *IPSN*, 2006.

[26] T. Malik, R. Burns, and N. Chawla, "A Black-Box Approach to Query Cardinality Estimation," in *CIDR*, 2007.

[27] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth Estimation: Metrics, Measurement Techniques, and Tools," *IEEE Network*, vol. 17, no. 6, pp. 27–35, 2003.

[28] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End-System Multicast," *IEEE J. Select. Areas Commun.*, vol. 20, no. 8, pp. 1456–1471, 2002.

[29] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-Aware Operator Placement for Stream-Processing Systems," in *ICDE*, 2006.

[30] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *ACM SIGCOMM*, 1998.

[31] The Netperf Benchmark. [Online]. Available: http://www.netperf.org/netperf/

[32] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The End-to-End Effects of Internet Path Selection," in *SIGCOMM*, 1999.

[33] N. Christofides, "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.

[34] J. Hoffer and D. Severance, "The Use of Cluster Analysis in Physical Database Design," in *VLDB*, 1975.

[35] P. W. Shor, "A New Proof of Cayley's Formula for Counting Labeled Trees," *J. Comb. Theory Ser. A*, vol. 71, no. 1, pp. 154–158, 1995.

[36] The Sloan Digital Sky Survey. [Online]. Available: http://www.sdss.org

[37] J. Gray and A. Szalay. (2003) Online Science: The World-Wide Telescope as a Prototype for the New Computational Science. Presentation at the *Supercomputing Conference*. [Online]. Available: http://research.microsoft.com/ Gray/JimGrayTalks.htm

[38] VOTable: An XML-based Encoding Scheme for Astronomical Tables and Catalogs. [Online]. Available: http://www.us-vo.org/VOTable/

[39] J. M. Hellerstein, M. Stonebraker, and R. Caccia, "Independent, Open Enterprise Data Integration," *IEEE Data Engineering Bulletin*, vol. 22, no. 1, pp. 43–49, 1999.

[40] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu, "Mariposa: A Wide-Area Distributed Database System," *VLDB Journal*, vol. 5, no. 1, pp. 48–63, 1996.