INVESTIGATING SKEW AND SCALABILITY IN PARALLEL JOINS

Christopher B. Walton

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188

TR-89-39 December 1989

Abstract

This research will improve understanding of the interaction between data skew and scalability in parallel join algorithms. Previous work in this area assumes that data are uniformly distributed, but data skew is widespread in existing databases.

This research makes three major contributions:

1. Several distinct types of skew are identified. Previous work treats skew as a homogeneous phenomenon, but simple analytic analysis shows that each type of skew has a different effect on response time.

2. The relative partition model of skew is defined. It is a simple analytic model that allows worst-case analysis of each type of data skew. The use of this model is demonstrated in an analysis of the sort-merge join algorithm.

3. A systematic plan for investigating skew and scalability. The interplay between simple analytic models and detailed simulations is vital: Analytic models bound the results expected from simulation, while more detailed simulation results validate the analytic models.

Keywords: skew, scalability, joins, performance, parallelism, database, multicomputers.

Contents

1 Introduction

Larger applications and cheaper hardware have generated great interest in parallel architectures and algorithms. Database processing is no exception: The join operation is important not only in traditional relational databases, but in logic and object-oriented systems as well. Furthermore, because the join is a computationally expensive operation, it is a good candidate for parallel implementation.

While there has been a great deal of research on parallel joins, nearly all of it assumes that data is uniformly distributed among processors at every stage of the join. Under these idealized conditions, join algorithms are quite scalable [11,13,19]. That is, performance increases approximately in proportion to the number of processors.

However, in more realistic situations, data skew will be present. At some point in the algorithm data will not be partitioned evenly between processors. The limited work done on data skew and join algorithms suggests that data skew can severely limit scalability. It is difficult to quantify this problem because existing characterizations of skew are very rough approximations.

Scalability of joins in the presence of skew has yet to be studied systematically. This report describes several advances in the study of this problem. More precise definitions of both skew and scalability are offered. Furthermore, several distinct types of skew are identified. The relative partition model, a simple analytic model of skew, is defined. This new model is also to demonstrate that each type of skew has different effects on performance. Finally, a methodology for investigating scalability of join algorithms in the presence of data skew is outlined.

This works draws on several areas within computer science. It is database research in that the join is a fundamental relational operation. Architectural issues must be considered as well. The concept of scalability comes from the study of parallel programming. This is also performance research in two ways: data skew is similar to service time variation; and performance measures, primarily response time, are used to connect skew and scalability. Figure 1 illustrates the intersection of these specialities.

The remainder of this report is organized as follows. Section 2 covers concepts in scalability, while section 3 provides background on data skew. Representative algorithms and architectures are covered in sections 4 and 5. respectively. Section 6 lists the research issues related to skew and scalability, and section 7 considers methods for investigating these problems. In section 8, the relative partition model is used to obtain best-case and worst-case estimates of response time for one join algorithm. These calculations also demonstrate how data skew increases response time. Finally, section 9 summarizes current research activity in the area of skew and scalability.

2 Scalability

The general topic of scalability encompasses two related sets of concerns: architecture and algorithms. Both are discussed below:

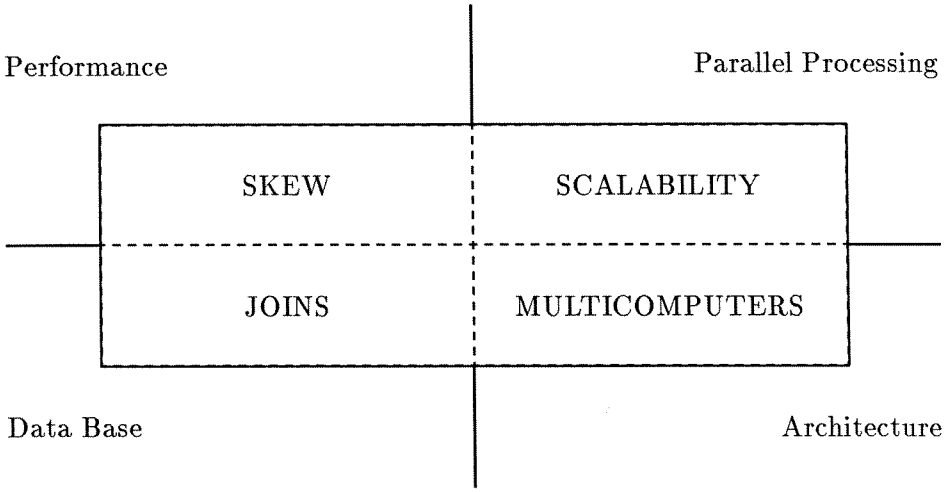

Figure 1: Problem Context

2.1 Architecture and Scalability

For a study of scalability, the primary independent variable is system size. Appropriate units for system size are essential if results are to be interpreted in a consistent and straightforward manner. Typically, system size is measured by counting CPU's. In numerical applications, this approach works well. However, the characteristics of relational processing are different, and counting CPU's is less satisfactory.

For join algorithms, disk performance is often a bottleneck, so the number of disks must be considered in computing system size. Another complication is that CPU's often vary in the amount of memory and type of peripherals attached to them. Performance can also be affected by auxiliary processors, such as communications and sort processors.

How should these disparate resources be mapped onto the single axis of system size? The approach taken here is to constrain how resources are added to the system. System size is measured in terms of *nodes*. A node may contain one or more processors, peripherals, and memory. That is, nodes constitute replicated groups of resources. The system architecture (nodes and communication subsystem) must be *inductive* in the sense defined by Lipovski and Malek[29]. The restriction to inductive architectures provides an unambiguous measure of system size, but still permits a wide range of configurations.

Inter-node communications is assumed to have little influence on scalability. This is a reasonable assumption because the communication subsystem is by far the least utilized resource in any system considered. Results of performance studies on the GAMMA[14] and Teradata [16] systems indicate that communications subsystem is never a bottleneck

2.2 Measuring Scalability for Algorithms

2.2.1 The Speedup Efficiency Metric

In discussing formal measures of scalability, we begin with the case where all steps in the computation can be run in parallel. Let n be the number of nodes, and let $T(n, K)$ represent the response time for some problem of size K on a system with n nodes. The problem size is fixed regardless of the value of n.

Scalability is frequently evaluated by calculating the speedup ratio R:

$$R(n, K) = \frac{T(1, K)}{T(n, K)} \tag{1}$$

The *ideal* response time is:

$$T_{ideal}(n, K) = \frac{T(1, K)}{n} \tag{2}$$

The ideal speedup ratio is:

$$R_{ideal}(n, K) = \frac{T_{ideal}(1, K)}{T_{ideal}(n, K)} = n \tag{3}$$

In practice, ideal speedup is never obtained. Intuitively, an implementation is scalable if the actual speedup is within a constant multiplicative factor of the ideal speedup. To make this idea quantitative, one can use the notion of speedup efficiency $\mathcal{E}$ (this presentation follows that of [8,7]).

$$\mathcal{E}_{fixed}(n, K) = \frac{R_{actual}(n, K)}{R_{ideal}(n, K)} = \frac{R_{actual}(n, K)}{n} \tag{4}$$

Note that $T_{ideal}(1, K) \equiv T_{actual}(1, K)$. Applying the definition of R_{actual}, one obtains:

$$\mathcal{E}_{fixed}(n, K) = \frac{T(1, K)}{nT_{actual}(n, K)} \tag{5}$$

Equation 5 defines speedup efficiency in terms of observable quantities.

Response time is an absolute measure that is affected by many factors. The same algorithm and the same inputs may be run on two computer systems and the response time will differ, due to differences in hardware performance, software overhead, communications bandwidth, and so on. In contrast, the speedup efficiency is a relative measure, with performance in the single node case serving as a reference point. Use of $\mathcal{E}$ factors out many of the implementation characteristics that affect response time. Thus, even if implementation differences between two algorithms preclude direct comparisons of response time, it is still possible to use $\mathcal{E}$ to compare their scalability.

2.2.2 Scalability Limited by Amdahl's Law

In most cases, a computation has some components that must be run on a single processor. According to Amdahl's Law [2], all computations contain a serial component, t_{serial} and a parallel component, $t_{parallel}$. That is:

$$T(1, K) = t_{serial} + t_{parallel} \tag{6}$$

The serial component imposes a lower bound on the response time, so that:

$$T(n, K) \geq t_{serial} + \frac{t_{parallel}}{n} \tag{7}$$

If this lower bound on response time is applied to the definition of speedup efficiency, one obtains:

$$\mathcal{E}_{fixed}(n, K) \leq \frac{t_{serial} + t_{parallel}}{nt_{serial} + t_{parallel}} \tag{8}$$

This equation implies that no algorithm is infinitely scalable: by raising n, the speedup efficiency $\mathcal{E}$ can be reduced below an arbitrary value.

2.2.3 Proportional Problem Size

Recently Gustafson and others [21] have advocated a different view of scalability. They argue that the size of a problem tends to expand as resources expand. That is, users will tolerate some range of response times, and will expand problem size to absorb additional processing power.

In Gustafson's model, the serial component is constant regardless of problem size, and the parallel component at each node is constant. Thus, for a problem of size nK, run on a single processor, the response time is:

$$T(1, nK) = t_{serial} + nt_{parallel} \tag{9}$$

The *scaled* speedup is:

$$S_{scaled} = \frac{T(1, nK)}{T(n, nK)} \leq \frac{t_{serial} + nt_{parallel}}{t_{serial} + t_{parallel}} \tag{10}$$

The speedup efficiency is then bounded by:

$$\mathcal{E}_{scaled} \leq \frac{t_{serial} + nt_{parallel}}{n(t_{serial} + t_{parallel})} \tag{11}$$

Under this formulation, it can be seen that the bound on $\mathcal{E}_{scaled}$ asymptotically approaches:

$$\frac{t_{parallel}}{t_{serial} + t_{parallel}} \tag{12}$$

From above. Also note that $S_{scaled} = n\mathcal{E}_{scaled}$.

2.2.4 Limitations of Fixed-Problem Size

Compared to the scaled problem size approach, the fixed-size method has two principal disadvantages. First, as the number of nodes increases, the total amount of memory in the system increases, so that a larger fraction of the input relations can be cached in memory. If join algorithms always exploit all available memory, then scaling up the system qualitatively changes the algorithm. This effect is seen in [11], where increasing the number of nodes increases total memory to the point where some disk I/O could be eliminated, resulting in a superlinear speedup. ($\mathcal{E} > 1.0$)

A second problem is that as the number of nodes increases, the problem size at each node may become trivially small. This exaggerates the impact of inter-node coordination and other overhead components.

$\mathcal{E}_{scaled}$ will be the primary metric used in this study. It represents a more realistic approach to scalability, and it avoids some of the complications inherent in the fixed problem approach.

3 Data Skew

3.1 Types of Data Skew

This section discusses the various ways data skew can arise. In the following discussion, a partition means the part of a (input or output) relation assigned to a node. If hashing is used, a partition may contain more than one hash bucket.

tuple population skew The initial distribution of tuples varies between partitions. For example, tuples may be partitioned by clustering attribute, in user specified ranges. Even if the populations are initially the same, differing rates of insertions and deletions may unbalance them over time. A mathematical model of this process is described in [22].

selectivity skew This occurs when the selectivity of local selection and projection predicates varies between nodes. A selection predicate that includes a range selection on the partitioning attribute is an obvious example.

hash function skew Hash function skew is a property of the hash function and not the data. It occurs when the number of possible hash key values mapped to each hash bucket varies. That is, the range of the hash function (bucket numbers) is less uniformly distributed than the range (hash key values).

join key skew This occurs when join key values are not uniformly distributed among tuples. For example: join key values follow a normal distribution. Join key skew is a property of data. If join skew is present, hash bucket sizes may vary even if there is no hash function skew.

3.2 Evidence for Data Skew

There is a variety of evidence for data skew. Skew has been observed in studies of existing data bases, such as [32,35,9]. It can be mathematically demonstrated that initially uniform data will probably become skewed by updates [22]. Corporate research and development groups are beginning to recognize the importance of skew, as shown by recent work on skew-resistant parallel algorithms for partitioning[24], sorting [31], and merging[45].

3.3 Models of Data Skew

Several options for modeling data skew are considered below. Let K be the size of a relation and N be the number of granules (or nodes). The sizes of individual granules are denoted by k_i, while k_{max} is the largest granule. Skew is quantified by the skew factor Q. The exact interpretation of Q will vary with the model and the type of skew.

The first two models presented here are suitable for analytic modeling.

3.3.1 The "absolute-partition" Model of Skew

Lakshimi and Yu [26,27] model skew by assuming that a fixed portion of a relation is assigned to one node. For simplicity, data are assumed to be uniformly distributed among the other $n-1$ nodes. That is,

$$k_{max} = Q \quad k_i = \frac{K-Q}{N-1} \tag{13}$$

where $1/n < Q < 1$.

This approach is not well-suited for a study of scalability. To see why, consider the ratio between the largest and smallest data partitions, r.

$$r = \frac{k_{max}}{k_i} = \frac{Q}{\frac{1-Q}{n-1}} = \frac{Q}{1-Q}(n-1) \tag{14}$$

Since Q is fixed, it can be seen that r increases proportionally with n. That is, in this model, data becomes more skewed as n increases. This is counterintuitive and makes it difficult to separate scalability limitations from the artificial increase in skew.

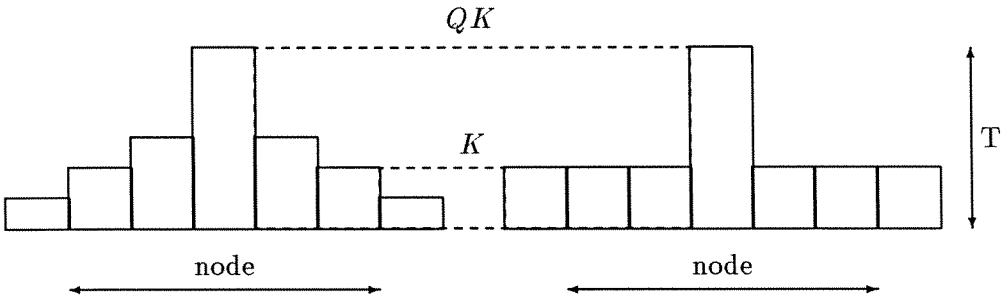

Figure 2: tuple population skew

3.3.2 The "relative partition" Model of data skew

To avoid the problems described above, the fraction Q can be defined in terms of the mean partition size. That is, let

$$k_{max} = Q\left(\frac{K}{N}\right) \quad k_i = K\frac{N-Q}{N-1} \tag{15}$$

Where $1 < Q < N$.

While the relative partition model may seem to be an excessive simplification, it is actually a reasonable approximation. This is due to the nature of join algorithms: there are multiple phases with barrier synchronization between phases. In many cases there is one data granule per node.

In a computation of this sort, the response time for each phase is determined by the Q-node – the one with the largest workload. Processing time for the other granules has little effect on the response time for each phase, since all nodes must wait for the Q-node to finish.

More formally, the response time T for a phase is:

$$T = f(K, Q)$$

Where f is a (possibly nonlinear) function of Q, the skew factor, and K, the per-node cardinality Increasing either variable will increase the response time. The skew factor Q expresses the extent to which the largest granule exceeds the average granule in processing time.

The relative partition model exploits this characteristic by focusing on the largest granule and treating all others as of uniform size. Figure 2 shows this simplification. The left side shows the actual distribution of granule sizes, while the right side depicts the simplified distribution used by the relative partition. Again, the model is accurate as long as the size of the largest granule is captured.

Figure 3 shows how the relative partition model can also model join key skew. In this case, skew is represented by assigning a larger join selectivity to the Q-node.

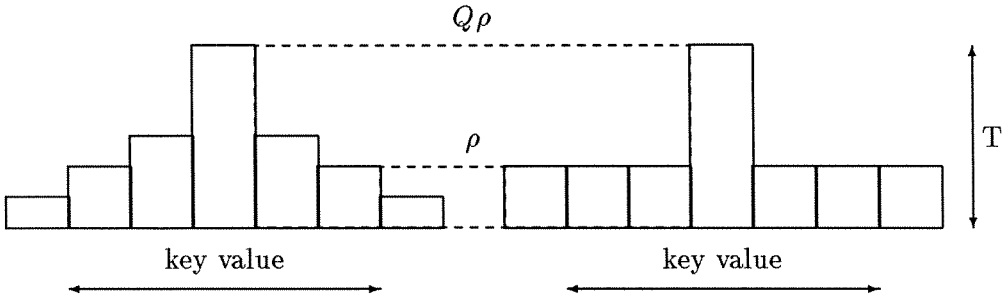

Figure 3: Join Key Skew

3.3.3 Modeling Data Skew with Random Variables

A more sophisticated approach is to model skew as a random variable. This leads to a more detailed model that is better implemented as a simulation. This approach allows one to draw on a considerable body of work on the characteristics of random distributions. See, for example, [28]. Also, many simulation packages support generation of random variables [39,40]

The interpretation of a random variable varies with the type of skew being modeled. Let X be a random variable. In a model of tuple population skew, X is the tuple population. For selectivity skew, X is the selectivity. For join key skew, X is the value of the join attribute.

In this investigation, three distributions will be used: uniform, normal, and Zipf. The merits of each are discussed below:

uniform The uniform distribution is often used when there is no empirical evidence to suggest anything else. The only parameters are the minimum and maximum values of the random variable. In the present case, it could be used to model tuple population skew or selectivity skew.

normal Many phenomena follow a normal distribution. A multivariate normal distribution is advocated in [9]. Haas[22] also makes a strong case for the normal distribution.

Zipf The Zipf distribution has been observed in a variety of disciplines[47,17,37]. The Zipf distribution corresponds to the "80-20 rule", the idea that 80 percent of the references are to 20 percent of the data.

3.4 Measures of Skew

It would be useful to have measures of skew that can be applied to any distribution. Two candidates are the variance and range.

range If samples from a random variable X_i represent the sizes of a group of data granules (such as hash buckets), then X_{max} is the largest granule. Let Q be the ratio between X_{max} and the average value of X_i.

Where there is only one data granule for each node, the response time for each node is determined by the granule size. Optimal response time (and utilization) are achieved when all granules are the same size – the no-skew case. As Q increases, the data become more skewed, response time increases, and utilization decreases. This situation corresponds to the relative partition model (see equation 15).

variance In cases where each node processes many granules, variance in granule size is a better measure of skew. It is a common result of queueing theory that response time increases with variance in service time, which would be proportional to granule size in most cases.

Both range and variance can be derived from the mathematical definition of all the skew models discussed above. They can also be estimated from observed data.

4 Parallel Join Algorithms

For a join to be efficient, processing of non-joinable tuples must be minimized. This is especially true in parallel applications, because of the added costs of transmitting tuples between nodes.

All the algorithms discussed here use a partioning hash function. As the first stage of join algorithms, both relations are hashed on the join attribute. Because the same hash function is used on both relations, a tuple will not join with any tuples outside of the bucket to which it is hashed. Furthermore, once the contents of each hash bucket have been delivered to the node that will perform the join, there is no need for inter-node communication until the tuples in each bucket are joined.

Examples of this class of join algorithms include sort-merge [38], hybrid hash (or GAMMA) [15], GRACE [25] and H-semijoin[11] Analytical studies by Richardson *et. al.* [36] indicate the performance advantages of the hash-based partitioning technique.

Filtering is another important technique for improving join performance. Either bit filtering [3,44] or semijoin filtering may be used to avoid transmitting and processing non-joinable tuples. Bit filters are relatively cheap to compute, but they do not eliminate all non-joinable tuples. Semijoin filters are computationally more expensive, but they do eliminate all non-joinable tuples.

5 Architectural Issues

Both architecture and algorithms affect scalability, so architecture cannot be ignored. In this research, multicomputer systems are of interest. These systems are composed of nodes that communicate exclusively by message passing. Each node has local CPU, memory and disk, and may

have auxiliary processors. This architecture is also known as a shared nothing architecture. The only shared resource is the interconnection network.

There are three motivations for this interest in multicomputers:

1. Most parallel join performance studies have been done on multicomputers.
2. Multicomputer architectures are more scalable than multiprocessors
3. Performance differences between multicomputers and multiprocessors are diminishing.

These arguments are discussed in more detail in [46].

There are several examples of systems used for parallel processing of database queries. For the most part, they function as "backends", receiving commands from a host system and returning results to it. Such systems include:

Teradata Teradata is described in [42,43,34,33]. Its performance is evaluated in [16].

GAMMA Gamma is an experimental system developed by the University of Wisconsin at Madison. It is described in [13,18]. Design alternatives for Gamma are evaluated in [19]. A benchmark is reported in [14], with a performance analysis in [12]. A second generation Gamma system is currently under construction [20].

ARBRE The Almaden Research Backend Relational Engine (ARBRE) project is described in [30]. Unlike the other systems described here, it distinguishes between virtual nodes (sites) and physical processors. This separation permits considerable configuration-independence in software design.

Bubba The goal of the Bubba project at MCC [1,10,6], is to produce a high performance database system from commodity parts by the early 1990's. Relations in Bubba are stored on a variable number of nodes, depending on various performance considerations[10]. A prototype Bubba system, the Experimental Vehicle (EV), has been constructed[41].

summary The systems presented above have many similar characteristics:

- All have architectures such that system size can be described in number of nodes.
- All use message passing for interprocess and internode communication.
- A minimum of specialized hardware is employed. The custom hardware is concentrated in the communications subsystem

- All systems can be incrementally scaled up by adding a few nodes (their architectures are inductive).
- Internode communication is not a performance bottle neck.
- Relations horizontally partitioned.

6 Research Issues

This research seeks to answer two related questions:

1. How does data skew limit join scalability?
2. What is the best way to preserve scalability when data are skewed?

6.1 Understanding Skew and Scalability

Developing a comprehensive theoretical framework for skew and scalability is difficult because it is a complicated and multifacted problem. There are many factors to consider, and few research results on the relative importance of these factors. While some first-order theoretical approximations can be made, a more empirical approach is needed to investigate the problem in detail.

In preliminary analytic analysis and subsequent experimentation, some of the issues to consider include:

- Are some algorithms less sensitive to skew than others?
- Does the type of the skew have any effect? There are preliminary indications (see section 8) that the effects of skew vary with the type of skew, but this area needs further study.
- How accurate are simple analytical models? It can be expected that assumptions of uniform data distribution will lead to optimistic performance and scalability estimates. Simple analytic models can also predict worst-case performance. However, very little is known about the accuracy of these results compared to those from more detailed models, such as simulation.
- What are the properties of the speedup efficiency $\mathcal{E}$, especially the relationship between $\mathcal{E}$ and parameters describing data skew (variance and range).

6.2 Dealing with Skew

In investigating ways to alleviate the effects of data skew, the most important questions are:

- are there algorithms insensitive to skew?

- when does it become worthwhile to dynamically redistribute data?
- can existing algorithms be modified, or are new ones needed?

7 Methods of Analysis

There are three major techniques for analyzing the performance of join algorithms. They are, in increasing difficulty of use, complexity analysis, analytic models, and simulation. While complexity analysis and analytic models can provide some valuable first-order results, the problem of skew and scalability is sufficiently complex that only simulation can provide the detail required to adequately analyze the problem.

7.1 Complexity Analysis

The first step in comparing algorithms is to determine their time complexity. Complexity analysis reveals the asymptotic growth of algorithm's time requirement as input size expands. If two algorithms have different asymptotic growth, the one with the lower exponent (slower growth) is preferred.

Basic time complexity analysis assumes an algorithm is executed sequentially on a single node. However, if one assumes that the algorithm has no serial components, then the operation count at each node of a N node system is roughly $1/N$ of the system total. That is, the complexity at the nodes is reduced by $O(N)$ in the parallel case. This provides an upper bound on the performance improvement that can be expected from implementing a parallel version of an algorithm.

Let $||R||$ be the cardinality of the larger relation R, and $||S||$ be the cardinality of the smaller relation S. Then the per-node cardinalities can be defined as $K_R \equiv ||R||/N$ and $K_S \equiv ||S||/N$.

Assume that there are at least N distinct join attribute values, and that these values are uniformly distributed among the tuples of both relations. That is, given V $(V > N)$ join attribute values, there are $||R||/V$ tuples with each value in R, and $||S||/V$ tuples in S. Since K_R and K_S would be $O(10^3)$ or larger in any realistic application, the requirement that each node have joinable tuples is certainly reasonable. This assumption excludes extreme cases where only a few nodes perform the entire join.

The naive nested loop is $O(NK_RK_S)$ while sort-merge and H-semijoin are $O(K_RK_S)$. (see [46] for a detailed complexity analysis of these algorithms) This demonstrates that while complexity analysis is useful for rejecting obviously inferior algorithms, it can not select between several promising ones.

Furthermore, complexity analysis cannot model important aspects of parallel join algorithms, such as skew, pipelining, and data dependent results (such as join selectivity)

7.2 Analytic Methods

Analytic methods can model pipelining and data dependent effects. Also, analytic models of the no-skew case and worst-case skew can provide performance bounds.

However, analytic methods have limitations. The most important is the need to make a number of simplifying assumptions; these assumptions preclude modeling data skew in full detail.

Analytic models typically make a number of *uniformity assumptions*. The principal ones are:

- Tuples are evenly distributed among nodes.
- Join key values are evenly distributed among tuples.
- All data granules are of uniform size

Adding data skew to the model violates these assumptions. The uniformity assumptions imply that all instances of a class of data granules (e.g: hash buckets) are identical. This interchangability substantially simplifies modeling. Under skew conditions, each granule is unique in size and other characteristics, such as distribution of join keys. Therefore, each granule and each node must be considered separately; the simplifications used in most analytic models can no longer be justified.

If analytic models are modified to incorporate skew, they tend to become less realistic in other respects. For example, Lakshimi and Yu [26,27] introduce a simple model of skew that does not include pipelining.

7.3 Simulation

For modeling skew effects at the level of detail needed for this study, simulation is the most practical method. With data skew, each data granule must be modeled explicitly. Nodes must also be modeled explicitly, since each has a different workload.

In principle, analytic methods could be used to obtain most of these results. However, the number of details and the complexity of the models employed make it more practical and less error prone to use a simulation to perform all the calculations. Furthermore, simulation software can automatically collect a variety of statistics to aid interpretation of the results.

The system simulated is deterministic. Once data granules, algorithm and architecture are defined, simulation will proceed without use of random variables. Granule characteristics can be defined in two ways. Profiles of data skew may be collected from real databases, or synthetic data may be generated using random distributions.

8 Example: The Sort-Merge Join Algorithm

This section uses the relative partition model of data skew to calculate response time of the sort-merge join algorithm in the presence of various types of data skew.

There are several motivations for showing these calculations in full. Beyond demonstrating the use of the relative partition model, (see section 3.3.2) these calculations show that skew does indeed degrade performance, and that each type of skew has different effects on performance.

8.1 Notation

Let the node with the larger number of tuples be the Q-node. Also, the larger hash bucket is called the Q-bucket.

Important dimensions in these calculations are:

N	The number of nodes.
R	The larger relation to be joined
$\|\|R\|\|$	denotes the cardinality of R
K_R	The number of tuples stored at each node $K_R \equiv \|\|R\|\|/N$
S	The smaller relation to be joined
$\|\|S\|\|$	the cardinality of S
K_S	is the per-node cardinality for S $K_S \equiv \|\|S\|\|/N$

The following constants represent time required for basic operations

t_{disk}	time to read or write a disk track
t_{msg}	time to send a message
t_{send}	time to send a message
t_{recv}	time to receive a message
t_{hash}	time to hash a tuple
t_{merge}	time to merge two tuples of the same relation
t_{join}	time to create semi-join or join output tuple
t_{scan}	time to scan an input tuple during merge-join

Other values used in the calculations include:

L_R	length of an R-tuple in bytes.
L_S	S-tuple length in bytes.
L_{key}	key length in bytes.
L_{tag}	tagged key length in bytes.
α_R	The local predicate selectivity for R.
α_S	The local predicate selectivity for S.
ρ_{join}	join selectivity
D	bytes per disk track
M	memory size in bytes
m	message size in bytes

In the relative partition model, each type of skew is represented as follows:

initial tuple population A disproportionate number of tuples are stored on one node. Specifically, there are QK tuples on one node, where K is the per-node cardinality.

predicate selectivity In this case, all nodes have the same initial population. However, during local selection (filtering), one node selects nodes with probability $Q\alpha$, where α is the average selection probability.

hash function skew Unless stated otherwise, the number of hash buckets is equal to the number of nodes. Tuples are Q times more likely to be hashed to the Q-bucket than to any other.

join key skew In this scenario, one partial join (one bucket) has a join selectivity of $Q_R Q_S \rho_{join}$, while the other partial joins have a selectivity of $N - Q_R Q_S \rho_{join}/N - 1$

8.2 The Sort-Merge Algorithm

For the purposes of this analysis, the sort-merge algorithm may be divided into three phases. Each phase must finish before the next can start. Within each phase, there are several steps; these steps can be overlapped (pipelined). The steps of the algorithm are:

1. Filter and Hash: The following procedure is performed for both the R-relation and the S-relation.
   (a) Read a track from disk
   (b) Apply local selection and projection predicates
   (c) calculate hash bucket
   (d) send to sort-merge site for that bucket (actually, tuples are collected in a buffer and sent once the buffer is full)

(e) store at sort-merge node. Write tuples to disk when they fill a track.

2. Sort Hash Buckets: Sort procedure is carried out for R and S separately
   (a) Retrieve tuples from disk until memory is full
   (b) sort tuples
   (c) Write sort run on disk.
3. Merge-Join
   (a) read a track from each run of R and S.
   (b) merge tuples from R tracks
   (c) merge tuples from S tracks
   (d) merge-join R and S tuples
   (e) write join output tuples to disk.

To simplify the analysis, two assumptions are made:

Assumption 1 *The tuples on each disk track can be filtered, hashed, transmitted to the join sites, and received there in less than the time required to read or write a track.*

Assumption 2 *Bit filtering is not used.*

8.3 Uniform Case

Filter and Hash Phase The time required to process a track at the storage node has three components: local filtering (selection and projection) , hash calculation, and transmission to join site. For the R relation, these sum to:

$$\frac{D}{L_R}t_{filter} + \frac{D}{L_R}\alpha_R t_{hash} + \left(\frac{N-1}{N}\right)\frac{D\alpha_R}{m}t_{send} \tag{16}$$

Similarly for S:

$$\frac{D}{L_S}t_{filter} + \frac{D}{L_S}\alpha_S t_{hash} + \left(\frac{N-1}{N}\right)\frac{D\alpha_S}{m}t_{send} \tag{17}$$

The time required to receive the tuples at the join site is

$$\left(\frac{N-1}{N}\right)\frac{D\alpha_R}{m}t_{recv} \tag{18}$$

for R, and:

$$\left(\frac{N-1}{N}\right)\frac{D\alpha_S}{m}t_{recv} \tag{19}$$

For S. Thus, assumption 1 can be stated more precisely as:

$$\begin{aligned} t_{disk} &> \frac{D}{L_R}t_{filter} + \frac{D}{L_R}\alpha_R t_{hash} + \left(\frac{N-1}{N}\right)\frac{D\alpha_R}{m}\left(t_{send} + t_{recv} \right) \\ \textbf{and} & \\ t_{disk} &> \frac{D}{L_S}t_{filter} + \frac{D}{L_S}\alpha_S t_{hash} + \left(\frac{N-1}{N}\right)\frac{D\alpha_S}{m}\left(t_{send} + t_{recv} \right) \end{aligned} \tag{20}$$

If the constraints in equation 20 hold, then all computation and communication can be overlapped with disk I/O, and the time required to complete the first phase is:

$$2\left(\frac{K_R L_R}{D} + \frac{K_S L_S}{D} \right) t_{disk} \tag{21}$$

Sort Phase In the general case, relation partitions on each node will be too large to sort in memory ($K_S > M$). Therefore, they must be divided into runs, each of which are sorted separately. Runs from both relations are then merged and merge-joined.

The number of sort runs for R is $K_R\alpha_R L_R/M$. Since there are M/K_R tuples per run, the time to sort each run is:

$$\frac{M}{L_R}\log_2\left(\frac{M}{L_R}\right)t_{sort} \tag{22}$$

The total sort time for R becomes:

$$K_R\alpha_R\log_2\left(\frac{M}{L_R}\right)t_{sort} + \frac{2K_R\alpha_R L_S}{D}t_{disk} \tag{23}$$

The last term in the equation reflects the time needed to read and write each tuple. Similarly, the time for S is:

$$K_S\alpha_S\log_2\left(\frac{M}{L_S}\right)t_{sort} + \frac{2K_S\alpha_S L_S}{D}t_{disk} \tag{24}$$

Merge-Join It is assumed that there is sufficient buffering so that no operation waits for the completion of a disk read. Each tuple must go through about $\log_2(K_R\alpha_R L_R/M)$ merge steps (comparisons). Thus, the time for the merge stage, including disk reads, is:

$$K_R\alpha_R\log_2\left(\frac{K_R\alpha_R L_R}{M}\right)t_{merge} + \frac{K_R\alpha_R L_R}{D}t_{disk} \tag{25}$$

The result for S is similar:

$$K_S\alpha_S \log_2\left(\frac{K_S\alpha_S L_R}{M}\right) t_{merge} + \frac{K_S\alpha_S L_S}{D} t_{disk} \tag{26}$$

Reading and scanning the input tuples for the merge join requires

$$K_R\alpha_R t_{scan} + K_S\alpha_S t_{scan} \tag{27}$$

The time required to write the joinable tuples is:

$$K_R\alpha_R K_S\alpha_S\rho_{join} t_{join} + \frac{K_R\alpha_R K_S\alpha_S\rho_{join}(L_S + L_R - L_{key})}{D} t_{disk} \tag{28}$$

Summing the results from equations 21, 23, 24, 25, 26, 27 and 28 yields the response time for the uniform case:

$$\begin{aligned} R_U \;=\; & \left((2+3\alpha_R)K_R L_R + (2+3\alpha_S)K_S L_S + K_R\alpha_R K_S\alpha_S\rho_{join}(L_R + L_S - L_{key}) \right) \frac{t_{disk}}{D} \\ & + \left(K_R\alpha_R \log_2\left(\frac{M}{L_R}\right) + K_S\alpha_S \log_2\left(\frac{M}{L_S}\right) \right) t_{sort} \\ & + \left(K_R\alpha_R \log_2\left(\frac{K_R\alpha_R L_R}{M}\right) + K_S\alpha_S \log_2\left(\frac{K_S\alpha_S L_S}{M}\right) \right) t_{merge} \\ & + (K_R\alpha_R + K_S\alpha_S) t_{scan} + K_R\alpha_R K_S\alpha_S\rho_{join} t_{join} \end{aligned} \tag{29}$$

8.4 Tuple population skew

In this case, response time increases because one node has to read more tuples from disk. The time for disk I/O during the filter and hash phase becomes:

$$2\left(\frac{Q_R K_R L_R}{D} + \frac{Q_S K_S L_S}{D} \right) t_{disk} \tag{30}$$

It is assumed that the extra tuples from the Q-node are uniformly distributed among to all join buckets. Therefore the sort and merge join phases have the same time requirements as the uniform case.
The response time then becomes:

$$\begin{aligned} R_{TPS} \;=\; & \left((2Q_R + 3\alpha_R)\frac{K_R L_R}{D} + (2Q_S + 3\alpha_S)\frac{K_S L_S}{D} \right) t_{disk} \\ & + \frac{K_R\alpha_R K_S\alpha_S\rho_{join}(L_R + L_S - L_{key})}{D} t_{disk} \\ & + \left(K_R\alpha_R \log_2\left(\frac{M}{L_R}\right) + K_S\alpha_S \log_2\left(\frac{M}{L_S}\right) \right) t_{sort} \end{aligned}$$

$$+\left(\ K_R\alpha_R\log_2\left(\frac{K_R\alpha_R L_R}{M}\right)+K_S\alpha_S\log_2\left(\frac{K_S\alpha_S L_S}{M}\right)\ \right)t_{merge}$$
$$+(K_R\alpha_R+K_S\alpha_S)t_{scan}+K_R\alpha_R K_S\alpha_S\rho_{join}t_{join} \qquad (31)$$

Note that the only change from equation 29 is in the first t_{disk} term.

8.5 Selectivity Skew

Under this scenario, each node has the same number of tuples stored on disk. However, on one node, tuple are Q times more likely to be selected by the local selection predicate. While the time to read and filter the tuples is the same, more time is required to hash and transmit tuples on the Q-node.

Selectivity skew will not increase response time unless the additional tuples on the Q-node cause assumption 1 to become invalid. In this case, the constraint expressed by equation 20 becomes:

$$\begin{aligned} t_{disk} &> \frac{D}{L_R}t_{filter}+\frac{Q_R D}{L_R}\alpha_R t_{hash}+\left(\frac{N-1}{N}\right)\frac{Q_R D\alpha_R}{m}\left(\ t_{send}+t_{recv}\ \right) \\ \textbf{and} & \\ t_{disk} &> \frac{D}{L_S}t_{filter}+\frac{Q_S D}{L_S}\alpha_S t_{hash}+\left(\frac{N-1}{N}\right)\frac{Q_S D\alpha_R}{m}\left(\ t_{send}+t_{recv}\ \right) \end{aligned} \qquad (32)$$

Again, it is assumed that the hash function distributes the extra tuples from the Q-node evenly among all hash buckets, so that the response times for the sort and merge-join phases are the same as the uniform case. Thus, as long the conditions in 32 hold, the response time is the same as the uniform case (see equation 29).

8.6 Hash Function Skew

In this case, skew occurs when a disproportionate number of tuples are hashed to one bucket, called the Q-bucket. The response time for the disk reading, filtering, hashing, and message passing steps are all the same as the uniform case. However, the sort and merge-join phases are delayed by processing the Q-bucket.

Sort Phase On the Q-node, the number of sort runs for R becomes $Q_R K_R\alpha_R L_R/M$. Since there are M/K_R tuples per run, the time for each run is:

$$\frac{M}{L_R}\log_2\left(\frac{M}{L_R}\right)t_{sort} \qquad (33)$$

Thus the total sort time for R becomes

$$Q_R K_R \alpha_R \log_2 \left(\frac{M}{L_R}\right) t_{sort} + \frac{2 Q_R K_R \alpha_S L_S}{D} t_{disk} \tag{34}$$

The last term in the equation reflects the time of reading and writing each tuple. Similarly, the sort time for S is:

$$Q_S K_S \alpha_S \log_2 \left(\frac{M}{L_S}\right) t_{sort} + \frac{2 K_S \alpha_S L_S}{D} t_{disk} \tag{35}$$

Merge-Join Each tuple must go through about $\log_2(Q_R K_R \alpha_R L_R / M)$ merge steps (comparisons). Thus, the time required to merge all the sort runs of R is:

$$Q_R K_R \alpha_R \log_2 \left(\frac{Q_R K_R \alpha_R L_R}{M}\right) t_{merge} \left(\frac{Q_R K_R \alpha_R L_R}{D}\right) t_{disk} \tag{36}$$

Where the last term represents the time need to read sort runs stored on disk. The result for S is similar:

$$Q_R K_S \alpha_S \log_2 \left(\frac{Q_S K_S \alpha_S L_S}{M}\right) t_{merge} + \left(\frac{Q_S K_S \alpha_S L_S}{D}\right) t_{disk} \tag{37}$$

Reading and scanning the input tuples for the merge join requires

$$Q_R K_R \alpha_R t_{scan} + Q_S K_S \alpha_S t_{scan} \tag{38}$$

The time needed to write the joinable tuples is:

$$Q_R K_R \alpha_R Q_S K_S \alpha_S \rho_{join} t_{join} + \frac{Q_R K_R \alpha_R Q_S K_S \alpha_S \rho_{join} (L_S + L_R - L_{key})}{D} t_{disk} \tag{39}$$

Summing the results from equations 21, 34, 35, 36, 37, 38 and 39 yields the response time:

$$\begin{aligned} R_{HFS} \;=\; & \left((2 + 3 Q_R \alpha_R) \frac{K_R L_R}{D} + (2 + 3 Q_S \alpha_S) \frac{K_S L_S}{D} \right) t_{disk} \\ & + \frac{Q_R Q_S K_R \alpha_R K_S \alpha_S \rho_{join} (L_R + L_S - L_{key})}{D} t_{disk} \\ & + \left(Q_R K_R \alpha_R \log_2 \left(\frac{M}{L_R}\right) + Q_S K_S \alpha_S \log_2 \left(\frac{M}{L_S}\right) \right) t_{sort} \\ & + \left(Q_R K_R \alpha_R \log_2 \left(\frac{Q_R K_R \alpha_R L_R}{M}\right) + Q_R K_S \alpha_S \log_2 \left(\frac{Q_S K_S \alpha_S L_S}{M}\right) \right) t_{merge} \\ & + (Q_R K_R \alpha_R + Q_S K_S \alpha_S) t_{scan} + Q_R Q_S K_R \alpha_R K_S \alpha_S \rho_{join} t_{join} \end{aligned} \tag{40}$$

8.7 Join Key Skew

In algorithms with hash partitioning, join key skew has almost the same effect as hash function skew. Join key skew occurs when some join key values occur more frequently than others. In the worst case, all the most common key values will hash to the same bucket (for both relations).

If there is no hash function skew, each bucket will have an equal number of key values. That is, the domain of the hash function will be evenly divided. Thus, the Q-bucket will hold more tuples than other buckets. In this case, all hash buckets contain the same number of tuples.

R-Tuples are Q_R times more likely to hash to the Q-bucket than any other. The Q-bucket will also receive Q_S times more S-tuples than other buckets. Thus, the response time for the sort phase is the same as for hash function skew. (see equations 34, 35, 36, and 37).

In the merge-join phase, the time to scan the tuples is also the same, as the hash function skew. (see equation 38)

The time to write the joinable tuples on the Q-node becomes:

$$Q_R Q_S K_R K_S \alpha_R \alpha_S \rho_{join} t_{join} + \frac{Q_R Q_S K_R \alpha_R K_S \alpha_S \rho_{join} (L_R + L_S - L_{key})}{D} t_{disk} \qquad (41)$$

The response time is obtained by adding equations 21, 34, 35, 36, 37, 38 and 41 to obtain:

$$\begin{aligned} R_{JKS} = & \left((2 + 3Q_R\alpha_R)\frac{K_R L_R}{D} + (2 + 3Q_S\alpha_S)\frac{K_S L_S}{D} \right) t_{disk} \\ & + \frac{Q_R Q_S K_R \alpha_R K_S \alpha_S \rho_{join} (L_R + L_S - L_{key})}{D} t_{disk} \\ & + \left(Q_R K_R \alpha_R \log_2 \left(\frac{M}{L_R}\right) + Q_S K_S \alpha_S \log_2 \left(\frac{M}{L_S}\right) \right) t_{sort} \\ & + \left(Q_R K_R \alpha_R \log_2 \left(\frac{Q_R K_R \alpha_R L_R}{M}\right) + Q_R K_S \alpha_S \log_2 \left(\frac{Q_S K_S \alpha_S L_S}{M}\right) \right) t_{merge} \\ & + (Q_R K_R \alpha_R + Q_S K_S \alpha_S) t_{scan} + Q_R Q_S K_R \alpha_R K_S \alpha_S \rho_{join} t_{join} \end{aligned} \qquad (42)$$

Note that equations 40 and 42 are identical.

8.8 Effect of Skew on Response Time

The amount of increase in response time varies with the type of skew. We can gain some understanding of this effect by fixing Q_R and Q_S and calculating the response time for each type of skew. This approach varies the *type* of skew for a fixed amount of skew. By subtracting the response time for the uniform case, the increase in response time for each skew type can be found.

Tuple Population Skew The increase for tuple population skew is found by subtracting equation 29 from equation 31:

$$R_{TPS} - R_U = \frac{2(Q_R - 1)K_R L_R}{D} t_{disk} + \frac{2(Q_S - 1)K_S L_S}{D} t_{disk} \quad (43)$$

Selectivity Skew As discussed previously, $R_{SS} = R_U$. (see section 8.5)

Hash Function Skew The total increase in response time for hash skew (from equations 29 and 40) is:

$$\begin{aligned} R_{HS} - R_U \;=\; & \frac{3\alpha_R(Q_R - 1)K_R L_R}{D} t_{disk} + \frac{3\alpha_S(Q_S - 1)K_S L_S}{D} t_{disk} \\ & + \frac{(Q_R Q_S - 1)K_R \alpha_R K_S \alpha_S \rho_{join}(L_R + L_S - L_{key})}{D} t_{disk} \\ & (Q_R - 1)K_R \alpha_R \log_2\left(\frac{M}{L_R}\right) t_{sort} + (Q_S - 1)K_S \alpha_S \log_2\left(\frac{M}{L_S}\right) t_{sort} \\ & + (Q_R - 1)\left(K_R \alpha_R \log_2\left(\frac{K_R \alpha_R L_R}{M}\right) \right) t_{merge} + Q_R K_R \alpha_R \log_2(Q_R) t_{merge} \\ & + (Q_S - 1)\left(K_S \alpha_S \log_2\left(\frac{K_S \alpha_S L_S}{M}\right) \right) t_{merge} + Q_S K_S \alpha_S \log_2(Q_S) t_{merge} \\ & + (Q_R - 1)K_R \alpha_R t_{scan} + (Q_S - 1)K_S \alpha_S t_{scan} \\ & + (Q_R Q_S - 1)K_R K_S \alpha_R \alpha_S \rho_{join} t_{join} \end{aligned} \quad (44)$$

In evaluating the increase in response time for hash function skew, non-linear effects are apparent. Both the merge and join output components of the response time increase by greater than linear factors. The $\log_2(Q_R)$ and $\log_2(Q_S)$ terms for the merge times are indications of this increase. The join output time increases by the product of the two skew factors.

Join Key Skew As discussed previously, $R_{JKS} = R_{HFS}$.

Conclusions The most fundamental result of this analysis is that different types of skew have different effects on response time. This becomes apparent when equations 43 and 44 are compared. This result calls into question the assumption that the source of skew can be ignored.

9 Summary

Data skew can degrade performance and scalability of parallel join algorithms. Current theory is deficient because it ignores data skew.

This report describes several contributions to research on this topic:

1. Systematic classification of skew types.

2. The relative partition model of data skew

3. An analytic demonstration that different types of skew have different effects on performance.

Work is in progress to examine skew and scalability. One goal of this work is to compare join algorithms under a uniform set of assumptions. In measuring scalability, the most important independent variable is N, the number of nodes. With respect to skew, the key variables are the source (type) of skew, distribution (model of skew), and the parameters describing that distribution. In all cases, the major dependent variable is $\mathcal{E}$, the speedup efficiency. The scaled problem size method will be used to evaluate scalability.

Research would proceed naturally in three stages. The first stage will establish a baseline of test cases with "no-skew" assumptions. Next, these same cases will be evaluated (by analytic and simulation methods) under skewed conditions. This will reveal the impact of skew. Finally, the algorithms under test will be modified to make them skew-resistant, and a third set of cases will be examined.

10 Acknowledgements

The author wishes to thank Alfred Dale, Roy Jenevein, Furman Haddix, Robert van der Geijn and Don Batory for valuable discussions.

References

[1] William Alexander and George Copeland. Process and dataflow control in distributed data-intensive systems. In *ACM SIGMOD International Conference on Management of Data*, pages 90–98, June 1988. [Chicago, IL, USA].

[2] G. Amdahl. Validity of the single-processor approach to achieving large-scale computer capabilities. In *AFIPS Conference*, pages 483–485, 1967. vol. 30.

[3] E. Babb. Implementing a relational database by means of specialized hardware. *ACM Transactions on Database Systems*, 4(1), March 1979.

[4] Chaitanya K. Baru and Ophir Frieder. Database operations in a cube-connected multicomputer system. *IEEE Transactions on Computers*, C-38(6):920–927, June 1989.

[5] Chaitanya K. Baru, Ophir Frieder, Dilip Dandlur, and Mark Segal. Join on a cube: Analysis, simulation, and implementation. In M. Ketsuregawa and H. Tanaka, editors, *Database Machines and Knowledge Base Machines*, pages 61–74, Norwell, MA, October 1987. Kluwer Academic Publishers. [Karuizawa, Nagano, Japan].

[6] Haran Boral. Parallelism and data management. In *Third International Conference on Data and Knowledge Bases*, June 1988. [Jerusalem, Israel].

[7] Luigi Brochard. Scalability, granularity, and parallelism. Research Report RC 14779, IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, July 1989.

[8] Luigi Brochard. Scalability, granularity, and parallelism of numerical algorithms. Research Report RC 14786, IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, June 1989.

[9] Stavros Christodoulakis. Estimating record selectivities. *Information Systems*, 8(2):105–115, 1983.

[10] George Copeland, William Alexander, Ellen Baughter, and Tom Keller. Data placement in bubba. In *SIGMOD Conference*, June 1988. [Chicago, IL, USA].

[11] Alfred G. Dale, F. Furman Haddix, Roy M. Jenevein, and Christopher B. Walton. Scalability of parallel joins on high performance multicomputers. Technical Report TR-89-17, University of Texas at Austin Department of Computer Sciences, Austin, TX, USA, June 1989.

[12] David D. DeWitt, S. Ghadeharizadeh, and Donovan Schneider. A performance analysis of the gamma database machine. In *SIGMOD Conference*, June 1988. [Chicago, IL, USA].

[13] David J. DeWitt et al. Gamma - a high performance backend database machine. In *Twelth Very Large Data Base Conference*, 1986. [Kyoto, Japan, August 1986].

[14] David J. DeWitt et al. A single user evaluation of the gamma database machine. Technical Report 712, University of Wisconsin Computer Sciences Department, Madison, WI, USA, August 1987.

[15] David J. DeWitt, R. Katz, F. Olken, D. Shapiro, Michael Stonebraker, and D. Wood. Implementation techniques for main memory database systems. In *SIGMOD Conference*, June 1984. [Boston, MA, USA].

[16] David J. DeWitt, Marc Smith, and Haran Boral. A single-user performance evaluation of the teradata database machine. Technical Report DB-081-87, MCC, Austin, TX, USA, March 1987.

[17] R. A. Fairthorne. Empirical hyperbolic distributions for bibliometric description and prediction. *Journal of Documentation*, 25(4):319–343, 1969.

[18] Robert H. Gerber. Dataflow query processing using multiprocessor hash-partitioned algorithms. Technical Report 672, University of Wisconsin Computer Sciences Department, Madison, WI, USA, October 1986.

[19] Robert H. Gerber and David J. DeWitt. The impact of hardware and software alternatives on the performance of the gamma database machine. Technical Report 708, University of Wisconsin Computer Sciences Department, Madison, WI, USA, July 1987.

[20] Shahram Ghandeharizadeh and David J. DeWitt. A multiuser performance analysis of alternative declustering strategies. Technical Report 855, University of Wisconsin Computer Sciences Department, Madison, WI, USA, June 1989.

[21] John L. Gustafson, Gary R. Montry, and Robert E. Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM Journal of Scientific and Statistical Computing*, 9(4):609–638, July 1988.

[22] Peter J. Haas. Workload imbalance and parallel processing. Reserach Report RJ 6936, IBM Almaden Research Center, San Jose, CA, USA, July 1989.

[23] R.-C. Hu and R. Muntz. Removing skew effect in join operations on parallel processors. Technical Report CSD-890027, University of California at Los Angeles Computer Science Department, Los Angeles, CA, June 1989.

[24] Balakrishna R. Iyer, Gary R. Ricard, and Peter J. Varman. An efficient percentile partitioning algorithm for parallel sorting. Research Report RJ 6954, IBM Almaden Research Center, San Jose, CA, USA, August 1989.

[25] M. Kitsuregawa, M. Nakano, and T. Moto-Oka. Application of hash to database machine and its architecture. *New Generation Computing*, 1(1), 1983.

[26] M. Seetha Lakshimi and Philip S. Yu. Effect of skew on join performances in parallel architecture. In *Symposium on Databases in Parallel and Distributed Systems*, pages 107–120, August 1988. [Austin, TX, USA].

[27] M. Seetha Lakshimi and Philip S. Yu. Limiting factors of join performance on parallel processors. In *Conference on Data Engineering*, pages 488–496, February 1989. [Los Angeles, CA, USA].

[28] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis.* McGraw-Hill Book Company, 1982.

[29] G. Jack Liposvski and Miroslaw Malek. *Parallel Computing: Theory and Comparisons.* John Wiley and Sons, 1987.

[30] R. Lorie, J. Daudenarate, G. Hallmark, J. Stamos, and H. Young. Adding intra-transaction parallelism to an existing dbms: Early experience. Technical Report RJ 6165, IBM Almaden Research Center, Almadem, March 1988.

[31] Raymond A. Lorie and Honesty C. Young. A low communication sort algorithm for a parallel database machine. Research Report RJ 6669, IBM Almaden Research Center, San Jose, CA, USA, February 1989.

[32] Michael V. Mannino, Paicheng Chu, and Thomas Sager. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3):191–221, September 1988.

[33] Philip M. Neches. Data processing system and methods. United States Patent 4,444,171.

[34] Philip M. Neches et al. Multiprocessor intercommunication system and method. United States Patent 4,412,285.

[35] G. Piatetsky-Shapiro and G. Connell. Accurate estimation of the number of tuples satisfying a condition. In *ACM SIGMOD Conference*, pages 256–276, June 1984. [Boston, MA, USA].

[36] James P. Richardson, Hongjun Lu, and Krishna Mikkilineni. Design and evaluation of parallel pipelined join algorithms. In *ACM SIGMOD Conference*, pages 399–409, May 1987. [San Francisco, CA, USA,].

[37] G. Scarrot. Will zipf join gauss? *New Scientist*, 62(898):402–404, 1962.

[38] Donovan A. Schneider and David J. DeWitt. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. Technical Report 836, University of Wisconsin Computer Sciences Department, Madison, WI, USA, April 1989.

[39] Scientific and Engineering Software, Inc., Austin, TX, USA. *SES/design User's Guide*, 1989. release 1.0.

[40] Scientific and Engineering Software, Inc., Austin,TX, USA. *SES/workbench Reference Manual*, April 1989. release 1.0.

[41] Marc Smith et al. An experiment on response time scalability in bubba. Technical Report ACA-ST-379-88, MCC, Austin, TX, USA, November 1988.

[42] Teradata Corporation. *DBC/1012 Database Computer Concepts and Facilities*, 1983. document number C02-0001-00.

[43] Teradata Corporation. *Teradata DBC/1012 Database Computer Systems Manual*, 1.3 edition, feb 1985. document number C10-0001-00.

[44] P. Valduriez and G. Gardarin. Join and semijoin algorithms for a multiprocessor database machine. *ACM Transactions on Database Systems*, 9(1), March 1984.

[45] Peter J. Varman, Balakrishna R. Iyer, and Donald J. Haderle. Parallel merge on an arbitrary number of processors. Research Report RJ 6632, IBM Almaden Research Center, San Jose, CA, USA, December 1988.

[46] Christopher B. Walton. Skew and scalability of parallel join algorithms on multicomputers. Ph. D. Proposal, Department of Computer Sciences, University of Texas at Austin, November 1989.

[47] G. K. Zipf. *Human Behavior and the Principles of Least Effort.* Addison-Wesley, Cambridge, MA, USA, 1949.