Modeling On-line Rebalancing with Priorities and Executing on Parallel Database Systems

Daniel C. Zilio

Abstract

Because changes to the database (DB) and workload occur during a DB system's lifetime, the physical DB design must evolve to sustain good performance. These changes are carried out by on-line reorganizations which access the DB and execute concurrently with the DB workload. Different performance intrusions are placed on the workload when a reorganization is assigned different priorities compared to the workload processes. Our work studies the effects of the reorganization priority-level on performance. There are two performance aspects of interest: the time taken for a reorganization and the intrusion the reorganization places on the workload. The reorganization we consider is a redistribution (rebalancing) of a relation across the nodes of a parallel shared-nothing system. A shared-nothing system contains multiple processors with their own local memory and disks, and these processors share only the interconnection network. We have seen that heavily loaded systems or highly imbalanced relations across the disks indicates the choice of a higher reorganization priority-level than the workload's. This paper also presents a metric and an analytic modeling method that can be useful for comparing reorganizations in general.

1 Introduction

In a database (DB) system, the physical DB design is chosen to allow the workload queries to obtain good performance. Some of the design components include data placement on a parallel system, indexes, and relation clustering on each system disk. Before the DB is created, the DB administrator (DBA) decides on the physical DB design, based on the initial input information available on the DB, the workload, and the system. However, as the workload, DB, and system change, or as more information of the inputs is obtained, the initial design decisions may begin to hinder the performance expectations of the workload queries. For example, the amount of records in a relation may differ across the disks causing the access times for each partition of a relation on a disk to differ which leads to load imbalance across the disks. When this data skew, or some other input information, is not taken into account during the physical DB design decision, the initial decision will result in a performance that may be less than that of a design achieved on greater knowledge of the inputs. To reap better performance for the workload, the physical DB design will need to be adjusted dynamically during the run-time of a DB system, e.g., creating new indexes or reclustering a relation. Adjustments are referred to as *reorganizations*, and a method for an adjustment is called a *reorganization strategy.*

Since many possibilities exist in a dynamic decision about a desirable reorganization, the complexity of the decision is high. To ensure that all pertinent facets of the decision are given appropriate weight, an automatic dynamic decision algorithm should be devised and used consistently. The devised algorithm must consider the method used to perform a reorganization, the expected impact of the reorganization on the workload, and the expected benefit to be reaped by the workload after the reorganization. Another aspect of the automatic algorithm is the availability of the DB during the reorganization. Today's DB systems must be available 24 hours a day 7 days a week, and hence a reorganization should allow the DB to be usable during its execution [8]. We call this type of reorganization an *on-line* reorganization. If the DB is made unavailable, the reorganization is called *off-line*.

We concentrate on reorganizations for parallel shared-nothing DB systems. A shared-nothing system is a parallel system where the only shared resource is the interconnection network, and each processor has its own local memory and disks. Each processor and its local memory and disks is called a workstation or *node*. One common type of reorganization is the rebalance of a relation across the nodes of a parallel shared-nothing DB system. Rebalancing is the movement of a relation's tuples (or records) between nodes, with the goal of balancing the number of tuples across the nodes. We study rebalancing as the reorganization type in this paper.

This paper focuses on a component required for the dynamic decision algorithm: the priority of the process that performs the on-line reorganization relative to the processes executing the workload. The priority of a reorganization strategy affects the impact or intrusion on the workload and the reorganization's response time. In our work, we consider three reorganization priority levels: higher, lower, or equal to the workload's level. A high priority for reorganization gets the reorganization done as quickly as possible, but disrupts regular workload processing; low priority for reorganization minimizes the impact on regular processing and provides greater availability of the system resources to the workload processes, but means that the reorganization may take a long time.

Our study shows that relative priority is an important factor for the decision algorithm to consider. We show that the level of reorganization priority with respect to the workload is affected by the arrival rate of the workload and the degree of imbalance before reorganization. The arrival rate of the workload causes the system to have a certain average load. We use arrival rate and load of the workload to mean the same things.

This paper also introduces a metric that can be used in a decision algorithm to compare reorganizations. We also present a method for analytically modeling the execution of the reorganization and workload processes to estimate the costs of each and the impact each has on the other.

The document is structured as follows. In Section 1.1, we present some related work and discuss why it is incomplete. A general and a specific reorganization problem are described in Section 2. Before the experiments are presented in Section 5, we first describe our reorganization comparison metric and the analytic modeling method, in Sections 3 and 4, respectively.

1.1 Related Work

Off-line reorganization studies for serial DB systems were some of the first to be done [11, 16, 20]. The authors were interested in determining the periodic times when a reorganization should be executed. Their assumption was that the workload cost and reorganization cost increased linearly based on the size of the DB. Therefore, the benefits of a reorganization were reaped up to the end of a DB's lifetime. They also only adjusted to one type of change - reclustering - and they only considered one off-line method of reclustering.

A reorganization method was given by Copeland et al. to re-decluster whole relations [7]. They briefly discussed the idea of comparing the benefits after a reorganization with the reorganization's cost so as to help guide the decision of whether to reorganize. However, they did not present a metric, and they only considered the case where a single reorganization strategy is chosen.

Some papers have dealt with load balancing where a small number of relation pages are moved from heavily-loaded nodes or disks to more lightly-loaded nodes [2, 19]. The algorithms used simple predefined strategies, and they did not factor in the cost of movement into their decision.

A few reorganization utilities have been described in the literature [18, 14, 15, 1, 12].

Each utility provides a fixed reorganization and strategy, such as index creation. All of these utilities would be possible candidates for a decision algorithm to consider. Therefore, a method should be created to automatically choose a utility and compare different utilities.

To compare different index-creation utilities, Srinivasan provided a *loss* metric [17]. This metric multiplies the loss of throughput during the reorganization by the reorganization's response time. As a result of this metric, longer reorganizations incur a greater *loss* value unless offset by a lower throughput reduction. This trade-off would be done when a reorganization is executed in a way that allows more concurrent users during its execution which means the reorganization will take longer to complete. Unfortunately, this metric cannot compare reorganizations yielding different DB states and benefits because it ignores the amount of benefit achieved after a reorganization. The significance of the benefit depends on how long the benefit lasts and on how long the reorganization takes. The metric also did not consider the importance of the need for a new DB design (or DB state) to improve the workload performance.

2 Problem Definition and Preliminaries

To give an indication of the importance of our work, we first present an overview of the general reorganization problem. We then present the specific problem solved in this paper, and we discuss the usefulness of our results in assisting the design of a general decision algorithm.

In a real system, changes to the DB, system, and workload occurs requiring the need for reorganization mechanisms and methods to choose among reorganization strategies. Changes to a DB's characteristics include imbalances of records across the nodes of a parallel system due to inserts, updates, and deletions in the DB. This imbalance will lead to an imbalance of computational activity for certain queries, such as selections across the nodes. New queries introduced into the system cause the performance requirements of the workload to change. The initial DB design decision may have been based on incomplete or estimated knowledge of the DB system, such as cardinality estimates. When new information becomes available, the design must reflect this current information.

To decide on how to best adjust to the changes, a general dynamic physical DB design decision is required. This decision can be stated as an optimization problem, as seen in Equation 1. The goal of the optimization is to maximize the *benefit* metric (defined in Section 3). The benefit metric measures the usefulness of performing a reorganization compared to not performing any reorganization, and thus this metric should be greater than zero for the reorganization we choose. To eliminate more possible reorganizations, we must ensure the performance of the new design, P_{new}, is better than that of the old, P_{old}. However, even when this constraint holds, the cost of performing the reorganization with the workload may exceed the performance gain of the workload after the reorganization. We also require that the reorganization completes, R, within a time interval, T (also defined in Section 3). Any other DB system constraints must also be satisfied. Examples of an other constraint includes a disk space limit and having an unsaturated and stable system where stability implies queue lengths at the resources do not grow indefinitely.

$$\begin{array}{ll} \max & Benefit\ Metric \\ s.t. & P_{old} < P_{new} \\ & Benefit\ Metric > 0 \\ & R < T \\ & Other\ constraints\ satisfied \end{array} \tag{1}$$

This optimization problem can be solved using a search algorithm. The search algorithm must compare different strategies in order to find the best on-line strategy.

However, a search algorithm comparing all reorganization possibilities has exponential complexity. For example, if we have R relations, each with A attributes, the number of possible index sets in the DB (excluding whether an index is clustered or not) would be $2^{\sum_{i=1}^{R} A}$. Given that our decision includes index selection and data placement, our algorithm has at least as much complexity as these individual decisions. They were shown to be

NP-complete [6, 5, 13] and must be solved using heuristics. The algorithm must also choose how a reorganization will execute, i.e., the execution plan of the reorganization, and how concurrent it will be with the workload processes. This extra information adds a greater number of search cases for the algorithm to consider, thus increasing the complexity.

In this paper, we limited the reorganization type to be a rebalancing of a single relation across a two node parallel shared-nothing DB system, as seen in Figure 1. The imbalance we consider is the imbalance of records or attribute values across the disks. This data imbalance (or data skew) results from a number of causes:

- Inserts, deletes, and updates of records in an imbalanced way across the nodes;
- Workload changes resulting in the access of records with specific attribute values to go to one of the nodes more than the other; and
- The addition of new nodes and disks which initially would contain no records.

Our rebalancing is the movement of the excess records of a relation on one node to a node with fewer of the relation's records. We use x to represent the degree of imbalance, such that a relation of R tuples has a proportion x ($0.5 < x \leq 1.0$) of the tuples on node 1 and $(1 - x)$ on node 2. The queries in our workload are file scans on the whole relation. These queries perform optimally when the relation is balanced, that is when they find half of the tuples at each node, since the node with more tuples determines the parallel execution times. Our reorganization strategy is as follows:

- Read in the $(x-0.5)R$ tuples to move from node 1;
- Send them to node 2;
- Insert them on node 2; and
- Delete them from node 1.

We assume that, as the reorganization is executing, the original tuples may still be accessed on node 1. The moved records would also only be accessible on the reorganization's completion.

Part of the rebalancing process's definition will be its priority related to the workload processes. We allow the reorganization process to have either higher, equal, or lower priority to the workload processes. The workload processes will all have the same priority. We study the different performance effects on the workload for the three different rebalancing priorities under various scenarios, such as differing workload arrival rates and degree of data imbalance across the nodes.

Even though we use a simple two node system, our work is useful for even more complex systems, workloads, and reorganization strategies. Our study can be applied when there is imbalance across more than two nodes, and when the DB has more than one relation. Our workload can also be changed to include different types of queries, such as exact-matches, smaller predicate-selection transactions, sorts, joins, etc. These queries are affected by the resource contention in an imbalanced state.

3 Benefit Metric

To compare different reorganizations, we designed a metric that includes the cost of performing the reorganization, the benefit a workload experiences after a reorganization is complete, and the time needed to develop (or switch to) the new DB state. This metric will be used by a dynamic decision algorithm to allow comparisons between reorganizations yielding the same or different DB states. We avoided using only the resulting throughput or the response time of the workload with the reorganized DB, because they only present the benefit after a reorganization.

Our metric is calculated using average response time curves, as shown in Figure 2. We estimate analytically the response times of the workload and the reorganization. On the x-axis, R is the reorganization's response time, T is the time interval; and on the y-axis, B, D and A are the average workload response times before, during, and after the reorganization, respectively. Note that the *before* response time is the same as that for not performing the reorganization. The metric is calculated as a relative benefit which signifies that choosing to exe-

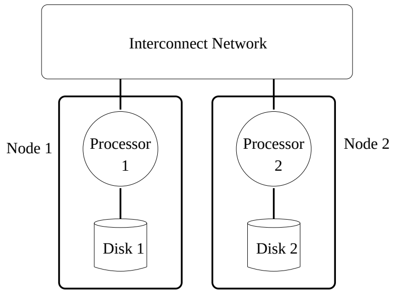

Figure 1: Two node shared-nothing system.

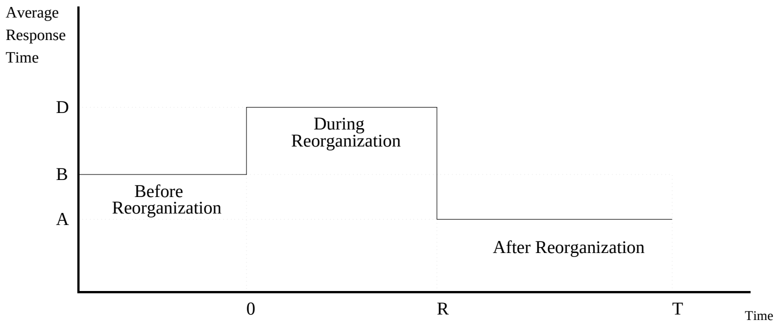

Figure 2: Example graph of a reorganization strategy's cost and benefit on a workload.

cute the reorganization (area under D between times 0 and R and under A between times R and T) leads to an overall performance gain compared to not performing the reorganization (the area under B between times 0 and T). A reorganization strategy is a good candidate in a decision algorithm if its relative benefit is greater than zero. During a reorganization's execution, the reorganization adds extra cost to the workload processes as seen by the area between response times D and B and times 0 and R, while after the reorganization completes at time R, the DB state should allow for a performance improvement by the workload, as seen in the area between response times B and A and times R and T. Because of the lack of future information, we assume our average workload response time estimates are steady throughout each time period. Our calculation thus determines the areas as rectangles. The relative benefit (or simply the benefit) metric is defined as follows:

$$\begin{aligned} \text{Benefit Metric} \quad &= P_{no\ reorg} - P_{reorg} \\ &= BT - (DR + A(T-R)) \\ &= (B-A)T - (D-A)R \end{aligned}$$

where P_i is the performance of the workload with a DB state which has or has not been reorganized. As can be seen, for a given workload and reorganization strategy, the benefit metric would be linear with respect to the time interval. This formula only makes sense when $R \leq T$; otherwise, a negative benefit results (assuming $A < B$).

We use response time so as to include the effects on all resources (in the form of added delays) when the reorganization and other queries are in the system. Using throughput to calculate the metric is limited because we viewed the queries as having given frequencies and thus would arrive with exponentially distributed interarrival times. The system must keep the throughput equal to these rates, otherwise it will become saturated over time [10], thus making the use of throughput uninteresting. If we were to detect a potential saturated system state, then any reorganization that moves the system to an unsaturated state is a good choice. For this paper, we assume the system is unsaturated.

To complete our benefit metric's explanation, we need to discuss the reason for limiting the time interval to some T. In reality, the benefit afterwards may not be reaped up to the end of the DB's lifetime. Other changes may happen in the future and the DBA may also want to limit how long the reorganization is allowed to have an impact on the system. For this reason, a time limit or *time interval*, T, is included in our metric to limit the amount of time in the future over which the metric is calculated.

There are different ways to determine an interval T. It is assumed this time is supplied by the DBA or some other source. Too long an interval is inappropriate because further system changes will occur. Too short an interval leaves insufficient time to reap a benefit, thus preventing any reorganization from being chosen. We could use a fixed time interval, or the time interval could be a weighted sum of N past time intervals where the initial N intervals would be arbitrary or supplied by an external source. A weighted sum allows the interval to adapt to recent past information. This recent past is used as an estimate of the near future.

4 Analytic Cost Estimation

To estimate the workload and reorganization response times, we use analytic methods. These methods are preferred because selecting a reorganization requires considering many possibilities, so the cost of each possibility must be estimated quickly. This criterion precludes the use of simulation and experimentation.

The method used to estimate costs is based on queuing network model (QNM) estimations. First, we require query plans as input so that we can determine the resources and access costs of a query. For each query and reorganization, we estimate the number of times each resource (CPU, I/O, communication) is visited (*visit count*). Some formulas are given in Appendix A. Once the visit counts are calculated, they are input into a QNM estimation engine to calculate how long each query and reorganization spends at a resource (*residence times*), including the delays caused by waiting for other queries or the reorganization process to use the resources. The residence time of resource k by a process j is represented by R_{kj}. Once the residence times are determined, the response time of a query or reorganization is calculated as: $R_{comm,j} + \max_i(R_{I/O_i,j} + R_{CPU_i,j})$, where $1 \leq i \leq n$ in an n node system. We needed to estimate the parallel execution costs separately from the QNM estimations because the QNM formulas sum up all the residence times to get the response times [10]. The average workload response time is then calculated as:

$$ART = \frac{\sum_{i=1}^{Q} f_i R_i}{\sum_{i=1}^{Q} f_i}$$

where Q is the number of different queries in the workload, f_i is the arrival rate of a query class, and R_i is the average response time of query i.

We model each resource as an M/M/1 queueing service center with preemptive scheduling [10, 9, 4]. The interconnection network is modeled as a single server where the communication speed is the time it takes to send a message over the network.

The reason for using such a modeling method is that the processes can be assigned priorities and we can use these priorities to model preemptive scheduling at the servers. This modeling allows us to determine added delays the reorganization places on the workload processes (and vice versa) when its priority is lower, equal, or higher to the workload's. For example, if a process enters the queue of a server using preemptive scheduling and the server is servicing a lower priority process, the server will preempt the low priority process and begin servicing the higher priority one immediately.

In our model, each query in a workload is represented as a transaction class with an arrival rate in the QNM, and the reorganization is represented by a batch class with one customer. A single customer batch class would begin executing on the system as soon as its previous execution was complete. Estimating its average performance would be the same as determining a single execution of it given an average delay from the rest of the customers. Since we want to determine the single execution of the reorganization with average external delays from the workload, then modeling it with a batch class is ideal. It should be noted that the *before* and *after* workload response times (B and A, respectively) are estimated without using the reorganization in the QNM formulas, and the workload's visit counts during a reorganization are calculated using the DB state before the reorganization begins. This before state is used since we assume the new DB state is not made available to the workload until a reorganization commits.

We used the approximate mean value analysis (AMVA) method to estimate costs with priorities and mixed-class workloads [9, 4]. The main formula representing the preemptive priority scheduling at a resource is:

$$R_{ks}(N) \;=\; \frac{V_{ks}[S_{ks}+\sum_{r\in H_s\cup E_s} S_{kr}A_{kr}^{(s)}(N)]}{1-\sum_{r\in H_s} S_{kr}\lambda_{kr}^{(s)}(N)}$$

$R_{ks}(N)$ is the residence time of class s at resource k when N customers are in the system; V_{ks} is the visit count for class s at resource k; S_{ks} is the resource cost per visit; H_s is the set of classes with higher priority than class s; E_s is the set of classes with equal priority to class s; $A_{kr}^{(s)}(N)$ is the number of customers at resource k for a class r when a customer of class s arrives; and $\lambda_{kr}^{(s)}(N)$ is the number of customers of class r that arrive while a customer of class s is in service. Note that A and λ are calculated using performance estimates from the other classes and the priority levels between classes.

The formula basically states that the residence time for a customer of class s is equal to its service time plus the increased times caused by higher- and equal-priority customers already in the server's queue (A) and ones arriving while this customer is queued (λ). The estimates of A and λ were done using the BKT method as described in Bryant et al. and Eager et al. [3, 9]. In this method, $A_{kr}^{(s)}(N) = Q_{kr}(N - e_s)$ with e_s is the unit vector with one at the s-th position and Q is the queue length at server k of customer type r, and $\lambda_{kr}^{(s)}(N) = X_{kr}(N)$ with X being the throughput at resource k. The visit count for a processor is the number of CPU instructions to execute. Service times S_{ks} (in milliseconds) for CPU, communication, and I/O are defined as $1/(MIPS \times 1000)$, $Commtime$, and $IOAccess$, respectively. The parameters are defined in Table 2, and some formulas using these parameters are given in Appendix A.

5 Experiments

In our experiments, we varied the arrival rates of the workload and the degree of imbalance, as defined in Sections 5.1 and 5.2, respectively. For a given load and degree, we determined the priority level that yields the best value for the benefit metric. The winning level is the one with the highest benefit for a given time interval, T. We noticed that as the arrival rate increases so does the best priority level. This result is due to the fact that the load at the resources increases with increasing arrival rate, thus making the data imbalance cause a greater load imbalance across the nodes which translates to higher query response times compared to a balanced system. This large performance difference causes a more urgent need for a reorganization's benefit to be realized so as to improve each query's performance which means the reorganization should be done at a higher priority. This fact also holds when the degree of imbalance increases. The load was varied by using different query arrival rates, and the degree of imbalance was varied through the variable x.

We use a set of inputs to determine the analytic costs of the reorganization and workload. Table 2 gives the system-related inputs, while Table 1 describes the inputs for the DB, workload, decision constraints, and reorganization.

Before presenting the experiments, we show in Figure 3 the response time curves of the workload and reorganization as the arrival rate varies in the system. The initial degree of imbalance for all experiments was 0.75 unless otherwise stated. We see that as the arrival rate increases, the impact that the reorganization and the workload have on each other causes the response times to grow until the system becomes saturated (arrival rate $>$ 0.0001 queries per millisecond or 0.1 queries per second). We plotted three different priority level response time curves for the reorganization and workload during a reorganization.

Figure 4 shows the benefit metric of all three

Parameter	Value	Description
R	100000	Number of tuples in relation
t	100 bytes	Tuple size
s	0.1	Selectivity of the queries on the relation
x	$0.5 < x \leq 1.0$	Proportion of records on node 1
T	Varies	Time interval
f	Varies	Arrival rate (queries/second) for query class

Table 1: DB, workload, reorganization, and decision constraint inputs.

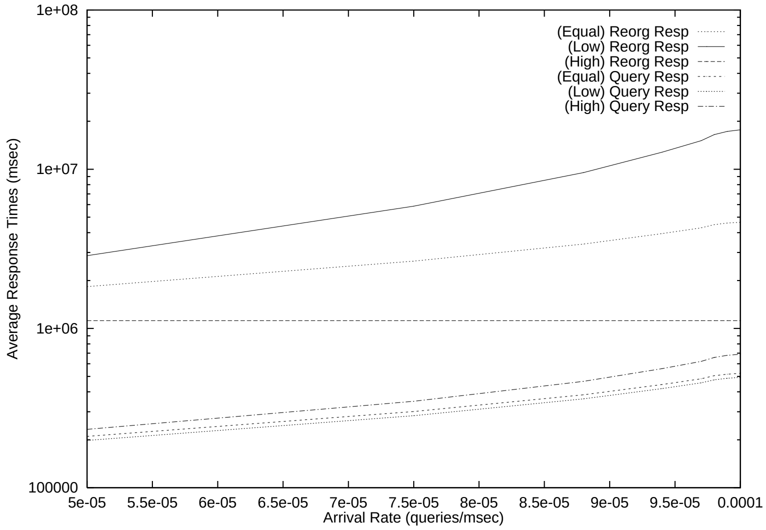

Figure 3: Sample graph of a query and reorganization response times.

priority levels when the time interval varies. Since the metric has little meaning when the time interval is less than the reorganization's response time, we change the formula to be $(B-D)T$ when $time < R$ to reflect the fact that the intrusion cost of the reorganization builds up over time. This formula is basically the negative area between the *before* and *during* workload response time curves. The curves use the benefit metric curve when $time \geq R$, so we see a change in slope for each curve at exactly the reorganization's response time. In other words, we changed the metric calculation to be:

$$\text{Benefit Metric} = \begin{cases} (B-A)T-(D-A)R, \\ \quad \text{if } R < T \\ \\ (B-D)T, \\ \quad \text{otherwise} \end{cases}$$

From these two figures, we see some phenomena that are due to modeling all the resources with preemptive priority scheduling. Since, the reorganization at low priority does not affect the workload response times during reorganization, the calculation of the curve during a reorganization is always zero in Figure 4. Also, at high priority, the reorganization response time is unaffected by the workload. We need to study the effects of changing the disks and network to be modeled using head-of-line (HOL) or first-come-first-serve (FCFS) scheduling in the future.

5.1 Varying the Load on the System

In these experiments, the degree of imbalance is fixed at $x = 0.75$. We found similar perfor-

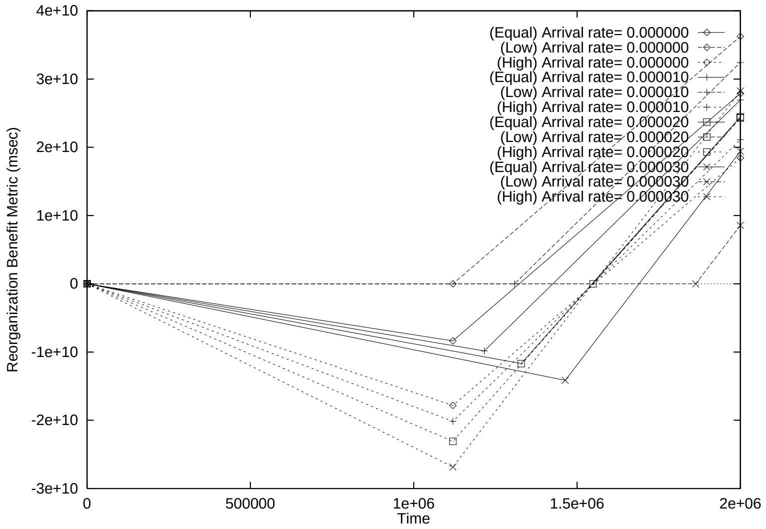

Figure 4: Benefit metric curves when time interval and arrival rates (in queries/millisecond) vary.

mance characteristics for other values of x. Our first set of experiments in Figure 4 shows different metric curves using the time interval as the x-axis. As the arrival rate increases, so does the time needed to reap a benefit after a reorganization increases according to benefit metric. The steepness of the curves also increases, indicating that the gain for high loads gives you greater payoffs for longer time intervals. Figure 4 shows that as the arrival rate increases, the winning priority level moves from low to equal to high.

The experiment in Figure 5 shows the performance trends when a reorganization starts to yield a benefit $\geq$ zero during the varying of the query arrival rate. We call this time the *break-even* point of the benefit metric. Different priority curves are shown, and the curve having the lowest value for a given arrival rate is the winner. There is a cross-over point from low to equal to high priority as the workload's arrival rate increases, and the load reaches a point where the need for a balanced DB state is greater than the cost for a higher priority reorganization.

5.2 Varying the Degree of Imbalance

In this section's experiments, we vary the degree of imbalance. The degree is used in the x-axis with different priority curves plotted to indicate the break-even point of the benefit metric. In Figure 6, we use an arrival rate of 0.00003 queries/millisecond. We see that when the degree of imbalance increases, the winning priority (the curve with the lowest time interval at an x value) moves from low to equal to high. However, we have seen that when the arrival rate is 0.00002, as seen in Figure 7, the cross-over point for the winning priority is at a higher degree of imbalance. This implies that at lower arrival rates, the need for reorganization is less urgent, which is consistent with the results of Section 5.1.

6 Conclusion

In this paper, we have shown that the priority level of a reorganization strategy is an important component for deciding how best to execute the strategy on-line with the workload of a parallel DB system. We have seen that this level is affected by the load the workload places on the system and by the degree of imbalance. We also presented a new metric to compare re-

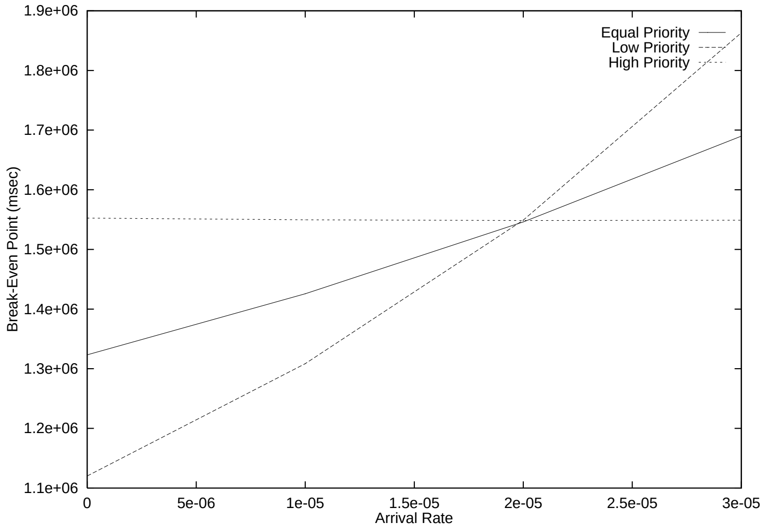

Figure 5: Arrival rate vs. break-even point where the break-even point is the time interval value when the benefit metric is zero.

organizations and an analytic method to estimate the execution of a reorganization strategy and the workload.

A result applicable to more complex systems and decisions is that the priority of reorganization affects the intrusion on the system, the cost-benefit metric result, and thus the possible strategy chosen by a decision algorithm. Solving the general decision algorithm is also assisted by our presentation of a benefit metric and our analytically modeling method.

We plan to extend this work in the future. More complex systems will be used. Larger system sizes and a greater number of relations will be studied, along with more reorganizations and strategies. One strategy would be to use indexes for a reorganization to allow faster selection of the records to redistribute, but doing so would lead to a greater insert and delete cost. The workload will include more varied and more complex queries, such as those involving updates. Updates increase the amount of intrusion, such as data contention, on the reorganization because both must update the relations. We will also unify our static physical DB design decision algorithm with a reorganization strategy selector to make a dynamic decision algorithm. This algorithm will include more than just rebalancing as the possible reorganizations.

Acknowledgements I would like to thank my supervisor, Ken Sevcik. His guidance on the QNM formulas, the idea of studying priority, and his support were greatly appreciated. Special appreciation also goes out to Kelly Lyons and Chaitan Baru for their assistance in developing the framework for this paper and in their helpfulness in bringing me into the Center for Advanced Studies (CAS) at IBM. This research was funded by CAS and NSERC.

About the Author Daniel Zilio is a PhD candidate at the University of Toronto in the Department of Computer Science. His thesis is titled "Physical Database Design Decision Algorithms and On-line Reorganization for Parallel Database Systems".

He can be reached at the address: *Department of Computer Science, University of Toronto*, Sanford Fleming Building, 10 King's College Road, Toronto, Ontario, Canada M5S 1A4. His Internet address is: zilio@cs.toronto.edu

A Visit Count Formulas

In this section, we describe the formulas

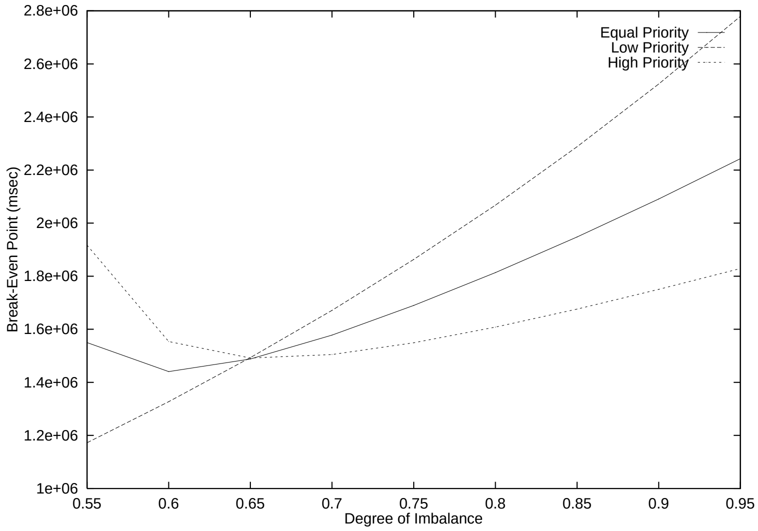

Figure 6: Degree of imbalance vs. break-even point where the arrival rate is 0.00003 queries/millisecond.

required to estimate the visit counts for the queries and reorganizations considered in this paper. We assume that the number of visits for a processor is equal to the number of processor instructions needed to execute the operator or command. We use the system parameters as defined in Table 2. R_i is the relation cardinality at node i, $pg(R_i)$ is the number of pages for a relation R on node i, and t is the tuple size.

The I/O visit count for reading R_i records on node i is:

$$V_{I/O} = pg(R_i)/Prefetch$$

The processor visit count for a selection on a table of R_i records is:

$$\begin{aligned} V_{CPU} &= pg(R_i) \times INST\_IO + \\ & R_i \times INST\_SELECT + \\ & INST\_START + INST\_STOP \end{aligned}$$

Visit counts for the communicating of R_i tuples would be calculated by:

$$V_{COMM} = R_i \times \lceil t/MaxMsgSize \rceil$$

Visit counts for inserting R_i records at node i is calculated as:

$$V_{I/O} = pg(R_i)$$

$$\begin{aligned} V_{CPU} &= pg(R_i) \times INST\_IO + \\ & R_i \times INST\_INSERT + \\ & INST\_START + INST\_STOP \end{aligned}$$

Finally, the visit counts for deleting R_i records at node i, which stores R records at that node in total is calculated by:

$$V_{I/O} = pg(R) + \min(R_i, pg(R))$$

Note that the I/O count represents the fact that we either delete records from all data pages or delete from as many pages as there are records to delete. The processor count is:

$$\begin{aligned} V_{CPU} &= V_{I/O} \times INST\_IO + \\ & R_i \times INST\_DELETE + \\ & INST\_START + INST\_STOP \end{aligned}$$

References

[1] C. K. Baru et al. DB2 Parallel Edition. *IBM Systems Journal*, pages 292–322, February 1995.

[2] Anna Brunstrom, Scott T. Leutenegger, and Rahul Simha. Experimental evaluation of dynamic data allocation strategies

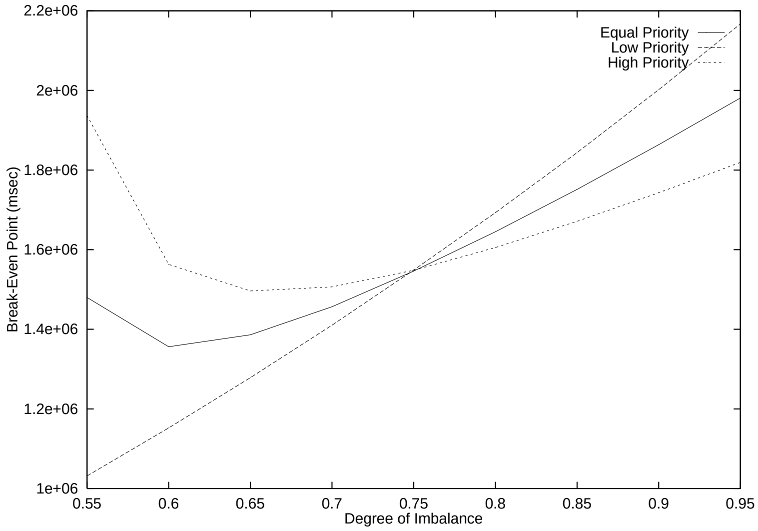

Figure 7: Degree of imbalance vs. break-even point where the arrival rate is 0.00002 queries/millisecond.

in a distributed database with changing workloads. Technical Report NASA Contractor Report 195024, ICASE Report No. 95-2, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, January 1995.

[3] R. M. Bryant, A. E. Krzesinski, and P. Teunissen. The mva pre-empt resume priority approximation. In *Proc. of the ACM SIGMETRICS Conf.*, pages 12–27, Minneapolis, MN, 1983.

[4] Raymond M. Bryant, Anthony E. Krzesinski, M. Seetha Lakshmi, , and K. Mani Chandy. The mva priority approximation. *ACM Trans. on Computer Systems*, pages 335–359, November 1984.

[5] Alberto Capara, Matteo Fischetti, and Dario Maio. Exact and approximate algorithms for the index selection problem in physical database design. *IEEE Transactions on Knowledge and Data Engineering*, pages 955–967, December 1995.

[6] Douglas Comer. The difficulty of optimum index selection. *ACM Trans. on Database Systems*, pages 440–445, December 1978.

[7] George Copeland, William Alexander, Ellen Boughter, and Tom Keller. Data placement in Bubba. In *Proc. of the ACM SIGMOD Conf.*, pages 99–108, Chicago, June 1988.

[8] David J. DeWitt and Jim Gray. Parallel database systems: The future of high performance database systems. *Comm. of the ACM*, pages 85–98, June 1992.

[9] Derek L. Eager and John N. Lipscomb. The amva priority approximation. *Performance Evaluation*, pages 173–193, 1988.

[10] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance.* Prentice-Hall, 1984.

[11] Guy M. Lohman and John A. Muckstadt. Optimal policy for batch operations: Backup, checkpointing, reorganization, and updating. *ACM Trans. on Database Systems*, pages 209–222, September 1977.

[12] Edward Omiecinski, Liehuey Lee, and Peter Scheuermann. Concurrent file reorganization for record clustering: A performance study. In *IEEE Int. Conf. on Data*

Parameter	Value	Description
MIPS	40	CPU MIPS rating
Prefetch	4	Number of pages prefetched in a disk read
IOAccess	16.43 msecs	Avg. disk access cost (read/write) per I/O request
Commtime	2 msecs	Communication cost per message
Pagesize	4 KB	Page size
MaxMsgSize	1 KB	Maximum message size
CPU Instruction Parameters		
INST_IO	10000	Instructions for I/O per page
INST_STARTCOMM	15000	CPU cost to send/receive a message
INST_COMM	20	Comm. cost per message byte
INST_SELECT	1000	Instructions per tuple per predicate
INST_INSERT	1000	Insert cost per tuple into base table
INST_DELETE	1000	Delete cost per tuple into base table
INST_START	5000	Start up operator cost
INST_STOP	5000	Terminate operator cost

Table 2: System inputs.

Engineering, pages 265–272, Tempe, Arizona, February 1992.

[13] Sriram Padmanabhan. *Data Placement in Shared-Nothing Parallel Database Systems.* PhD thesis, EECS Dept., Univ. of Michigan-Ann Arbor, 1992.

[14] Betty Salzberg and Allyn Dimock. Principles of transaction-based on-line reorganization. In *Proc. of the Int. Conf. on VLDB*, Vancouver, 1993.

[15] Gary S. Smith. Online reorganization of key-sequenced tables and files. Technical Report Tandem Systems Review, Tandem Computers Inc., October 1990.

[16] Gary H. Sockut and Robert P. Goldberg. Database reorganization – principles and practice. *ACM Computing Surveys*, pages 371–395, December 1979.

[17] V. Srinivasan. *On-Line Processing in Large-Scale Transaction Systems.* PhD thesis, Dept. of Comp. Science, Univ. of Wisconsin-Madison, Madison, WI, 1992.

[18] V. Srinivasan and Michael J. Carey. Performance of on-line index construction algorithms. Technical Report TR-1047, Dept. of Comp. Science, Univ. of Wisconsin-Madison, Madison, WI, September 1991.

[19] Gerhard Weikum, Peter Zabback, and Peter Scheuermann. Dynamic file allocation in disk arrays. In *Proc. of the ACM SIGMOD Conf.*, pages 406–415, Denver, May 1991.

[20] S. B. Yao, K. S. Das, and T. J. Teorey. A dynamic database reorganization algorithm. *ACM Trans. on Database Systems*, pages 159–174, June 1976.