Data reorganization in parallel database systems

Appears in *Proc. of the IEEE Workshop on Advances in Parallel and Distributed Systems*, Princeton, NJ, pp.102-107, Oct. 1993.

Chaitanya Baru

IBM Toronto Lab
IBM Canada Ltd.
North York, Ontario M3C 1W3

Daniel C. Zilio

Department of Computer Science
University of Toronto
Toronto, Ontario M5S 1A4

Abstract

Parallel database systems are suitable for use in applications with high capacity and high performance and availability requirements. The trend in such systems is to provide efficient on-line capability for performing various system administration functions such as, index creation and maintenance, backup/restore, reorganization, and gathering of statistics. For some of these functions, the on-line capability can be efficiently supported by the use of "incremental algorithms", i.e., algorithms that achieve the function in several, relatively small (i.e., less time-consuming) steps, rather than in a single, large step. Incremental algorithms ensure that only small parts of the database become inaccessible for short durations as opposed to non-incremental algorithms which may lock large portions of the database or the entire database for a longer duration. In this paper, we discuss issues in providing concurrent data reorganization capability using incremental algorithms in parallel database systems.

1 Introduction

Dynamic database reorganization algorithms have been studied in the past, in the context of single-node (or *sequential*) database systems. A survey of these issues is provided in [4, 6]. The focus of these studies has been on determining reorganization intervals for a database. More recently, some work has also been done on dynamic reorganization at the file level [5]. The advent of parallel database systems motivates us to study database reorganization in such systems. In this paper, we identify issues in dynamic database reorganization in shared-nothing parallel database systems. In the following subsections, we provide some background on reorganization for load balancing and the different situations which may lead to reorganization. In Section 2, we describe the various issues in reorganization using the example of a single node system. Next, in Section 3, we introduce our model of a parallel database system and discuss reorganization issues in such systems.

1.1 Reorganization and Load Balancing

The primary motivation for data reorganization is the improvement in performance obtained by a more balanced loading of the computer system. In the past, the issue of load balancing has been addressed in parallel computer systems in terms of scheduling jobs at different nodes in the system with the emphasis being on compute intensive rather than I/O or data intensive tasks. The scheduling of tasks may be performed *statically*, i.e. when a job first starts, or *dynamically*, i.e. during the course of execution of a job. Dynamic scheduling also implies that there is no natural "home", or site, for performing the respective tasks. In a database system, tasks correspond to the various operations in a query tree which operate on base data. It is more cost effective to schedule such tasks/operations at the nodes that contain the input data. However, operations that use intermediate and not base data as inputs, could be scheduled at any node in the system. Thus, in a parallel database system one can use both static and dynamic scheduling. The load at a node thus consists of the statically and dynamically scheduled operations at that node. Database reorganization can be used to change the loading due to both types of operations and also to change operations from being dynamically scheduled to being statically scheduled, and vice versa. While reorganization can be an expensive operation, the payoff for reorganization can be large due to the repetitive nature of queries in typical operating environments.

1.2 Types of data reorganization

We classify the situations under which reorganization (reorg) becomes necessary into *planned* and *crisis* situations. In a planned situation, the user is able to anticipate the need for reorg and take appropriate actions in advance. The actions can be taken before system performance falls below a certain threshold. In a crisis situation, the need for reorg is discovered only after the performance has fallen below a threshold. Thus, neither the system nor the user was able to predict the need for reorg. A planned situation may arise when the system has reliable means for predicting database growth and/or change in performance or when the user is able to anticipate a future change in workload or database size, e.g., a planned discount offering of tickets by an airline. In this case, the database is reorganized to provide better throughput for certain anticipated transactions. A crisis situation may arise due to the inability of the system to predict the future performance trend or due to a sudden change in the workload or system environment, e.g., failure of a node in the parallel computer system.

2 Reorganization in a Single Node System

A database reorganization typically consists of the following activities, (1) gather and analyze run-time statistics to decide on the need for reorg, (2) estimate performance before and after reorg, (3) estimate the cost of the reorganization phases, including the increased cost of normal database operations due to reorg, along with any resource usage by the reorganization process. Past work has studied the problem of determining the interval between consecutive reorganizations in order to optimize the total operating cost in a single node database system. In Yao [6], he considers both a constant cost as well as a linearly growing cost function with respect to execution time of the reorg operation. The reorg operation and other database operations are executed in a serial manner, i.e. the reorg operation locks out all other database operations. In our study, we are interested in determining intervals between reorgs as well as in reorganization strategies where the operation is performed concurrently with the execution of the regular database workload. An incremental approach is employed in order to minimize the effects on the database workload. Rather than locking the entire database (or, even one or more tables), only small portions of the data are locked during reorg.

First, we discuss issues in reorganization using the example of a single node database system. In figure 1, the x-axis is time and the y-axis is the performance metric of the workload. We use the weighted average response time of all queries in the workload as the metric. A parallel database system supports intra-query and intra-operator parallelism to provide improved response times for complex queries. Thus, response time is a useful measure of system performance. The weighted average response time is used in order to account for the different frequency of execution of different queries in the workload. The database structure is assumed to change over time due to insert and update operations. As a result, the average query response time deteriorates (increases) with time. This phenomenon is illustrated in figure 1 by the line marked, *Ideal Performance*. This ideal performance is obtained using continuous reorganization of the database. The rate of response time deterioration is say, θ_1. In practice, due to a variety of skews and non-uniformities in the growth of the database and changes in the workload, the actual performance may deteriorate more rapidly than the ideal. Thus, without any reorganization, the actual rate of increase of response time is say, θ_2, where $\theta_2 > \theta_1$.

The jagged edge curve in figure 1 denotes the query performance when a concurrent, incremental reorganization algorithm is employed. The reorganization is initiated when the performance reaches the point A on the performance curve. The purpose of the reorganization is to obtain a system performance that is closer to the ideal performance line (say, point I in figure 1). Each step of the incremental algorithm is represented by an upward line in the jagged curve. The first increment is accomplished between points A and B, the second between C and D, and so on. After each increment, the portion of the database that has been reorganized becomes available and, as a result of the reorganization of this segment of the database, there is an improvement in the overall performance (line segments BC and DE, in the figure). It is possible to stop incremental reorganization after only a few iterations. In this case, the overall query performance may improve, but it is still worse than the ideal. The advantage, however, is that one does not incur the cost of reorganizing the entire database.

The concurrent reorganization initially decreases system performance since, as soon as the reorg is started, there is an overall increase in load in the system due to increased data and resource contention. This is denoted by θ_3 in figure 1. The effect of each step of the incremental, concurrent reorg operation is indicated by the slopes of the lines, AB, CD, EF, and GH in figure 1. After the i-th increment, the response time decreases by ΔR_i. The time interval of each step is denoted by, t_I. The current reorg phase starts at time, t_S, and the next reorg phase starts at t'_S. The time, t_E, denotes the end of the previous reorg phase and t'_E denotes the end of the current reorg. The interval, $t_{SR} = t_S - t_E$, denotes the time elapsed since the last reorganization. The reorg step takes $t_r = n \times t_I$ time units, where n is the number of increments. In each step of the incremental algorithm, the intrusion of the reorganization degrades performance by an extra rate of say, θ_4, giving a resultant reorg rate of, $\theta_3 = \theta_2 + \theta_4$. The average response time is assumed to increase at a rate of θ'_2 after the first increment.

The effect of each step in the reorg operation is shown more clearly in figure 2. Assuming linear functions for performance, the costs and benefits of each increment, and of the entire reorganization phase, can be determined by computing the areas of the triangles above the lines indicating actual performance (which have rates of θ_2, θ'_2, etc.) and the trapezoid which stretches to the time t'_S. For each increment, the appropriate performance line is considered, e.g. AB for the first increment, CD for the second, etc. For the first increment, $Cost_i = Area\ of\ ABX$, and $Benefit_i = Area\ of\ XYZC$. For the entire reorganization phase, the utility of performing a reorg is computed as, $U = \sum_{i=1}^{n}(Benefit_i - Cost_i)$, where i is the increment index.

To obtain the reorganization strategy with the best benefit-cost difference, we need to determine appropriate values for θ_3, n, and t_I. The value of θ_3 is based on the type of reorganization algorithms chosen. As part of our future work, we plan to characterize reorg costs for different reorg algorithms and use them in the analysis to determine values for n and t_I. Finally, the analysis presented here is equally applicable for planned or crisis reorganization situations. The difference in the two cases is the point in time at which the reorg phase is started.

3 Reorganization in Parallel Databases

The parallel database system model we assume fits the *shared-nothing* architecture in which a set of independent computers (nodes) are connected via an interconnection network. Each node has its own disk and memory. The database is partitioned across the nodes of the parallel computer system. The query plans generated by the parallel database system take advantage of the database partitioning by incorporating intra-query and operator-level parallelism in the plans. The single node reorg analysis of the previous section can now be extended to the parallel database case in the following manner. First, in addition to monitoring the system level peformance, the performance of each node is also monitored by tracking the response times of the operations and/or subqueries executing at that node. The initial data placement provides a certain performance at the system level, which translates to a certain performance characteristic at each node. The performance characteristic of each node may be different. Over time, as the database partitions at each node undergo different rates of updates and inserts and as the characteristics of the workload change, there may be a deterioration in system performance (i.e. increase in response times). When the system performance falls below a threshold, a reorganization phase is initiated at all nodes. At this point, the node level performance is examined to determine the strategy for database reorganization.

Figure 3 shows the relevant segment of the performance graphs of nodes in a two node parallel database system. When the decision is made to reorganize, Node 1 is performing worse than Node 2, with respect to their expected performance. The reorg operation moves data from Node 1 to Node 2. In case 1, there is a net improvement in the performance of Node 1 (the performance at the end of reorg is at C) while the performance of Node 2 becomes worse (performance is at C'). Also, as shown in the figure, it is possible that after the reorg, the performance at Node 2 degrades faster (with rate δ_2^{new}) than before (δ_2'), and at Node 1 it degrades slower than before (θ_2^{new} vs. θ_2'). In this example, Node 1 sees a net benefit from reorg whereas Node 2 sees a net cost. However, the net effect is an improvement in the overall system performance. Alternatively, the reorg operation may include performing a node-level reorg (see Sec. 3.2) at Node 2 if this results in improved performance at Node 2. This is illustrated in Case 2 of figure 3.

As before, the cost of reorg at each node is determined by the areas of the triangles for each increment, while the benefit is determined by the area of the trapezoid between the old and new degradation curves up to the start of the next reorg, viz. t'_S. Note that the degradation line ending at point Z' may exceed the previous point Y', in case 2 of figure 3, thus incurring some costs rather than benefits at the end of the trapezoid.

3.1 Reorganization costs

To arrive at appropriate cost models for reorganization, we need a model for data placement in the parallel database system. The initial data placement problem for parallel database systems has been addressed in [1, 2]. Here, we assume that a set of tables is partitioned into groups based on their affinity, i.e. frequency of inter-table operations. Each group is then mapped to a cluster of processors in the parallel system using techniques similar to those mentioned in [2]. For example, Case 1 of figure 4 shows three groups with Group X containing tables R, S, T, U; Group Y containing tables P, V; and Group Z containing tables W, Q.

The clustering and initial placement decisions are based on the input workload information. The workload includes both unary (e.g., selects and aggregations) and binary (e.g., joins) operations. The binary operations provide a logical link between tables. In assigning groups of tables to a group, we choose tables that have strong links, i.e. the frequency of inter-table operations is high. Appropriate partitioning strategies are chosen for tables within a group, in order to obtain the best response time performance. While the workload may still contain inter-group queries, the grouping of tables is meant to minimize their contributions to the overall response time. The cluster of processors assigned to each group may either be disjoint or overlapping. The choice of the partitioning strategy, the number of processors assigned to a group, and the decision to use overlapped or non-overlapped clusters of processors are the issues addressed by the initial placement.

Changes in database size and query frequencies make the initial placement suboptimal and, therefore, cause a degradation in performance. This gives rise to the need for a reorg. To aid in the reorg decision, in addition to monitoring performance at the system level, there is also a need to monitor the performance (response time) at each cluster of processors and also at each node. The system-level information is used to initiate the reorg process. The cluster-level information and node-level performance information is then used to decide on specific reorg strategies. Cluster-level information can also be used to initiate a reorg within a cluster. Thus, there are three levels of reorg possible in a shared nothing parallel database system: *system-level, cluster-level, and node-level.* These are described further in the next section.

3.2 Types of reorg

A node-level reorg is performed locally at each node in the system, and is used to restructure the data physically stored at the node, including creating new indexes and views. Thus, there is no transfer of data among nodes. For this type of reorg, the analysis is similar to the single node case in figure 1. Node-level reorg effects the performance of queries that run at that node alone. However, there may not be much effect on queries that span more than one node. Performing only node-level reorganizations may not be sufficient to obtain the best system performance. Thus, it may become necessary to transfer data and associated queries between clusters of nodes.

A cluster-level reorg is used to improve performance of queries that execute within a cluster. This type of reorg requires moving data between the nodes of the cluster. Cluster-level reorg can include operations such as adjusting the partitioning of tables, reassigning tables across nodes, changing the partitioning key of a table, and adding a new node to the cluster.

A system-level reorg attempts to balance the performance across clusters. There are several possible reorg actions that can be taken at the system level, e.g. redefining groups of tables and remapping groups to clusters. The regrouping of tables can result in inter-cluster queries becoming intra-cluster queries and vice versa. This type of reorg can significantly effect system response time. The response time of intra-cluster queries is effected by remapping groups of tables to clusters, since the actual nodes used and the number of nodes assigned to these queries may change. When a node is reassigned to a new group of tables, the reorg must first move all existing tables from this node and then move data from the new group of tables to the node.

In a given reorg phase, any combination of the three reorg strategies can be selected. The net benefit provided by a reorg strategy depends on the cost incurred in doing the reorg and the resulting system performance. For example, while the operations involved in a system-level reorg are more complex, it is possible that a node-level or cluster-level reorg is more expensive than a system-level reorg. As part of our future work, we plan to characterize the cost/benefit of each level of reorg.

4 Summary

The goal of our reorganization study is to find the number of reorg intervals, the time of the intervals, the reorg algorithm and intrusion costs, and the cost-benefit differences at each node which maximize the benefit of the reorg across the system. In a parallel database system, we can address the reorg issues at the node, cluster, and system levels. Each level has a certain cost and benefit associated with it. In deciding on when to start the reorg, we need to determine the load imbalance or threshold level that must be exceeded to warrant the cost of a reorg algorithm. This decision process is periodically executed in the system.

Our future work consists of incorporating realistic cost formulas for reorg and analyzing graphs similar to the one in figure 1, in order to determine values for parameters such as the rates of performance detrioration, number of reorg intervals, etc. Another issue that we plan to study is the gathering of relevant statistics to aid in reorganization. For example, it may be possible to characterize the response time performance using other system statistics that can be collected with lesser overheads. In [3], they mention the use of a "load index" which could include CPU utilizations, CPU queue lengths, etc. as an indicator of response time behavior. We are interested in using a similar concept in the context of parallel database systems. A third issue to study is the effect of reorg on inter-cluster queries.

In several parallel architectures, e.g. hypercubes, meshes, hierarchical multiprocessor systems, the notion of clusters mentioned in this paper maps naturally to a cluster of processors in the system. Mapping groups of tables to physical clusters can potentially reduce query response times due to lower latency in inter-processor communications. We are interested in studying the effects of mapping groups of tables to these physical clusters.

References

[1] Copeland, G., Alexander, W., Boughter, E., and Keller, T., "Data placement in BUBBA," *ACM SIGMOD-88*, Chicago, June 1988.

[2] Padmanabhan, S. and Baru, C., "Data placement in parallel database systems," *ACM Intl. Conf. on Information and Knowledge Management*, Nov. 1993, Baltimore, MD.

[3] Shivaratri, N. G., Krueger, P., Singhal, M., "Load distributing for locally distributed systems," *IEEE Computer*, Dec. 1992.

[4] Sockut, G. H., Goldberg, R. P., "Database reorganization - Principles and practice," *ACM Computing Surveys*, Dec., 1979.

[5] Weikum, G., Zabback, P., Scheuermann, P., "Dynamic file allocation in disk arrays," *ACM SIGMOD-91*, May 1991.

[6] Yao, S. B., "A dynamic reorganization algorithm," *ACM Transactions on Database Systems*, June 1976.

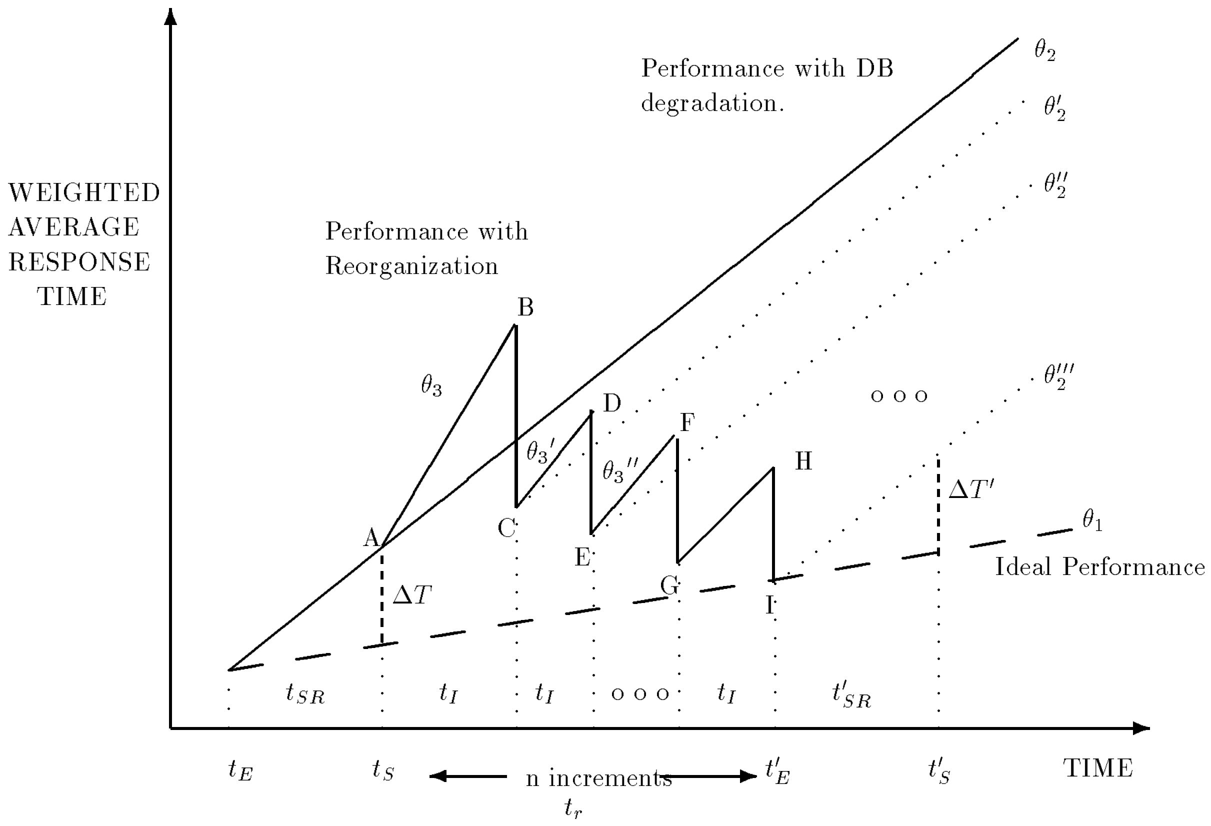

Figure 1: Local reorganization performance with parameters. Performance curves indicate the effects of incremental reorg on query execution time. (The jagged edge curve denotes the performance when an incremental, concurrent reorganization is executed with other DB operations.)

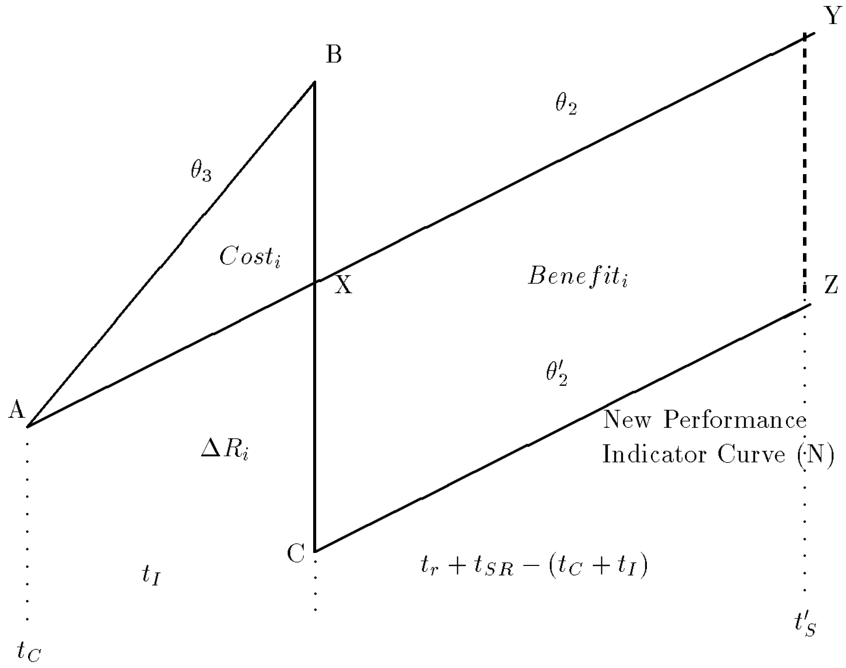

Figure 2: One increment of the local reorganization algorithm. The area of the triangle ABX denotes the cost, while the area of the trapezoid XYZC denotes the benefit. t_C is the time at which the increment begins executing.

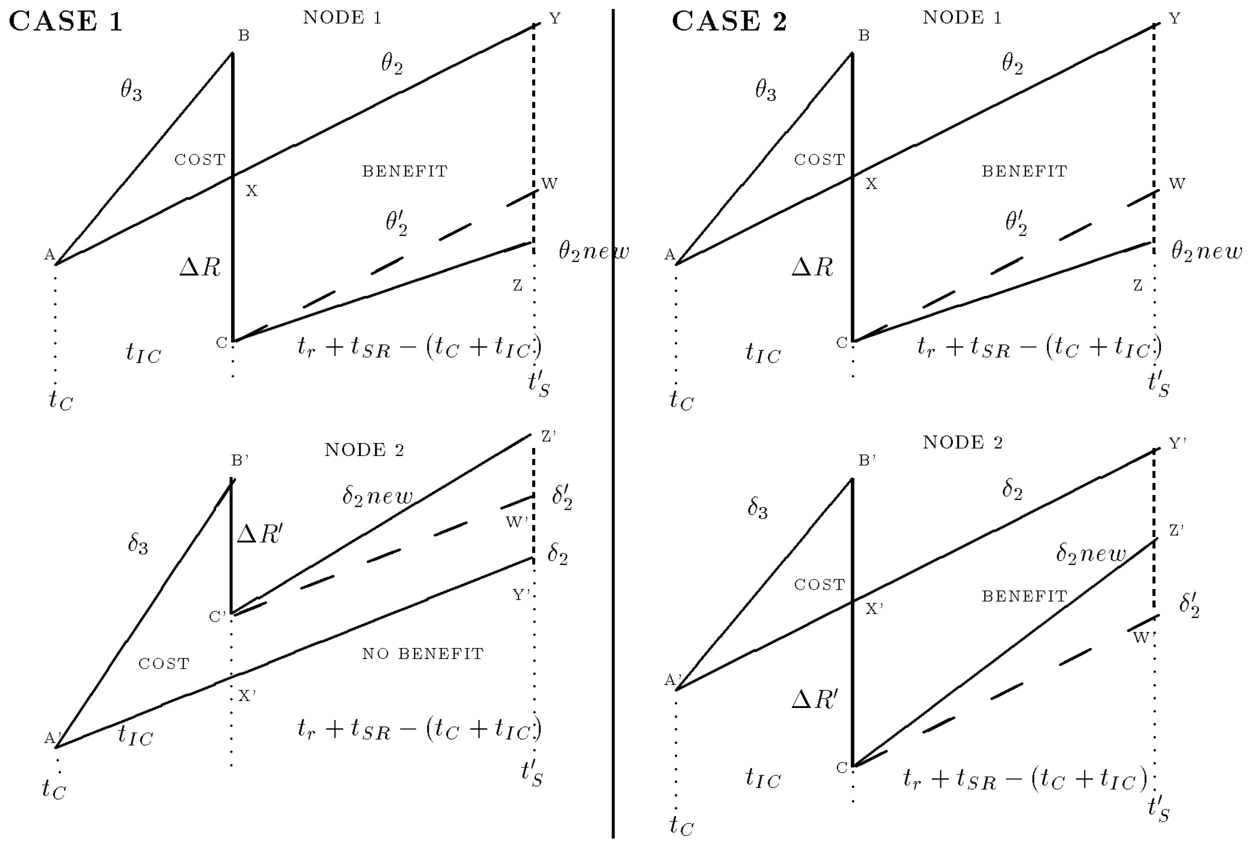

Figure 3: Cases 1 and 2 illustrate the possible parallel reorganization situations. Each case represents one increment in the reorganization. In both cases, Node 1 moves data to Node 2 potentially resulting in a slower rate of deterioration at Node 1, θ_2^{new}, and an increased rate for Node 2, δ_2^{new}.

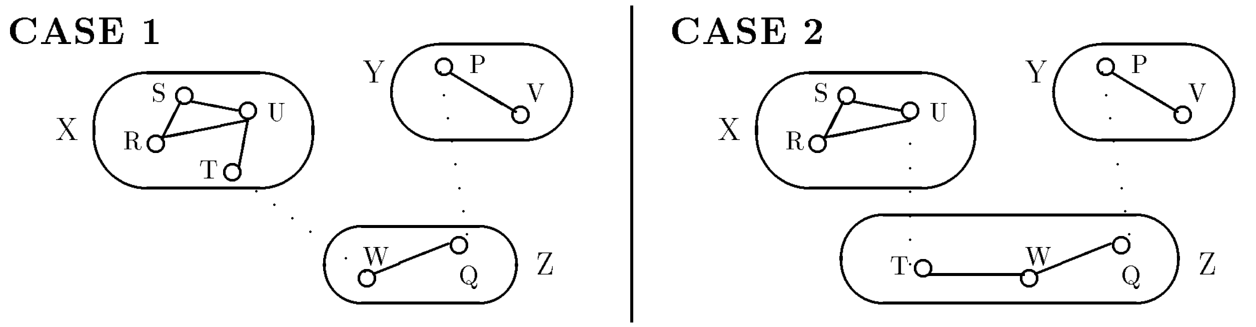

Figure 4: Example of a clustered placement. *Case 1:* Relations R,S,T, and U are placed in group X; P, V in group Y; and W, Q in group Z. *Case 2* demonstrates a possible reorganization. Due to the increased execution frequency of the query linking relations T and W in case 1 (denoted by the dotted line TW), relation T is moved over to group Z in case 2.