Using Simulated Annealing to Guide Server Data Placement

Juan Rubio and Lizy Kurian John
Department of Electrical and Computer Engineering
The University of Texas at Austin
{*jrubio,ljohn*}*@ece.utexas.edu*

Abstract

Modern commercial applications access large volumes of data, typically spread among several storage locations. These locations may be in different disks belonging to clustered or tightly coupled multiprocessor servers. In order to achieve good performance, data needs to be carefully distributed.

We propose a technique that uses simulated annealing to help with the placement of data in a server system. Simulated annealing, very popularly used in VLSI design automation tools, is seen to be useful in deciding the initial data placement as well as run-time placement of data as its usage evolves.

The proposed technique is applied to a group of queries from a decision support system (DSS) benchmark suite. It is observed that improved static placement using simulated annealing can produce speedups of up to 13% when data placement is globally optimized for all queries at once. When tuning the system to a single query we see speedups of up to 22%. The simulated annealing technique can also be applied over a query at run-time to improve the data placement from the initial globally optimized data layout. Such a dynamic reorganization is seen to result in speedups that match the speedups of query-specific data layouts. Simulated annealing is seen to be an effective way to reach near-optimal data layouts in a quick manner.

1 Introduction

Modern commercial applications access large volumes of data, typically spread over several storage modules. The servers that run these commercial applications typically have multiple processing elements, whether clusters of computers or tightly coupled multiprocessors. Figure 1 shows a sample multi-tier architecture, like those used in conventional transaction processing systems.

The multiple storage locations, which may be located in different nodes within the system, are not all equally accessible by the processing elements. Local accesses are usually

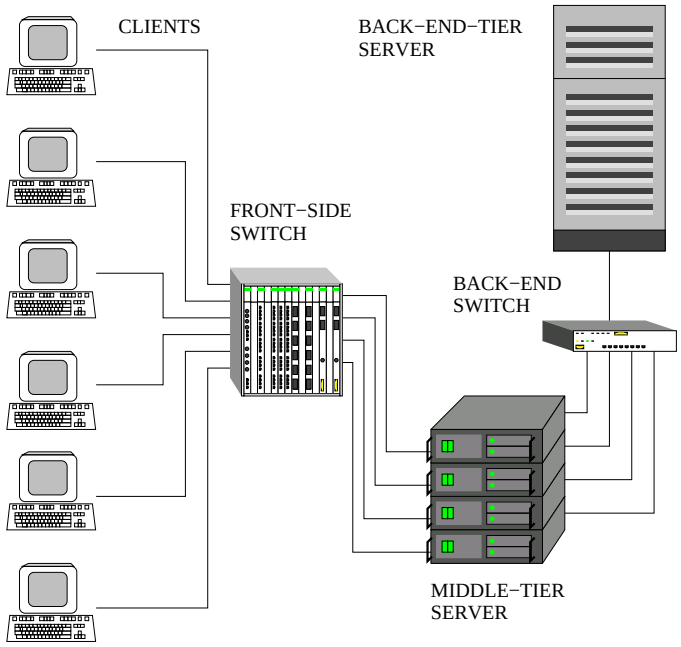

Figure 1: A conventional multi-tier system architecture used in a transaction processing system.

fast; however, remote accesses incur an extra access latency due to (i) the communication hardware and (ii) contentions in the global interconnect.

To improve the performance of servers on emerging workloads, the latency for data accesses must be reduced. One important factor to consider is the placement of data; an efficient placement minimizes data transfer at run-time. However, it is also difficult to arrive at an optimal data layout because ideal placement for one query may not be ideal for another query. Furthermore, it is many times difficult to get *a priori* knowledge of all queries. Even *a priori* knowledge of all queries does not lead to ideal performance, because as we show, layouts optimized statically over all the queries are not optimal. The best performance can be obtained using globally optimized initial placement followed by quick and low-overhead dynamic data reordering. For this we need adequate heuristics to quickly and efficiently reorganize the data at run-time.

This paper shows how principles of combinatorial analysis and simulated annealing can be used to improve server data placement. *Combinatorial analysis* is the branch of mathematics that studies the different possible arrangements of discrete objects. Combinatorial optimizations are used in many branches of sciences to find the best arrangement of objects for a given constraint. For example, in the context of VLSI and Electronic Design Automation, combinatorial optimizations are commonly used to solve placement and route problems [1, 2]. These problems involve determining the best location to place modules in the chip to

reduce its total area, or the way to route interconnect lines within or between the modules. In this paper we show that similar techniques can be used to aid server data placement.

We demonstrate how the data partitioning problem in a server can be expressed in terms of the minimum cut (min-cut) problem studied in combinatorial analysis. We propose the use of simulated annealing to guide the placement of data in the system. An efficient initial data layout for a group of operations (queries) is first identified. We then present a dynamic data reorganization technique for adapting to the needs of a changing workload. This technique uses of a set of profiling counters to assist the data fetching engines and query scheduler in the run-time reorganization of the data.

We explore the effectiveness of simulated annealing for data placement using operations from a decision support system (DSS) benchmark. The TPC-R [3] benchmark suite is used to test the initial data placement and then the dynamic data reorganization technique is evaluated using operations from the TPC-H [4] benchmark. The initial placement using simulated annealing principles results in speedups of up to 13% when the optimization is performed over all the queries (i.e., global optimization). Tuning specifically to each individual query results in a speedup of up to 22%. Starting from the globally optimized data layout, dynamic data reorganization using simulated annealing is able to approach the query-specific data layout in approximately 10 annealing steps. Here we see speedups of up to 31%, which exceeds the speedups obtained with the ideal query-specific training set. This is due to the ability of the technique to tune the layout to the different phases of the query. The sensitivity of the results to the different parameters of the technique is also evaluated.

The rest of the paper is organized as follows. Section 2 introduces the basics of the simulated annealing technique. Section 3 explains how the server data placement problem can be formulated into a graph and how simulated annealing can be applied to find an ideal initial placement. Section 4 presents the run-time data reorganization technique. Section 5 describes our performance evaluation methodology. Section 6 presents and analyzes the results of our evaluation. Section 7 presents the conclusions of this work.

2 Simulated Annealing

This paper addresses the placement of data in the different nodes of a server. The data may be accessed by processors in local or remote nodes. Our main goal is to reduce the number of remote data accesses performed by those processors. The task of selecting the combination that results in the lowest cost has been shown to be NP-complete for general graphs [5]. In practice, it is not easy to find an adequate partition for more complex configurations, so most algorithms use heuristics to approximate a solution.

Simulated Annealing (SA) [6, 7] is a technique that allows us to explore the design space of possible combinations in a quick and effective manner. It identifies a combination that would result in a value for the objective function near its global minimum. It does not guarantee the optimal solution, but given adequate parameters it produces a close solution in a fraction of the time.

As its name implies, the SA technique is modeled on the annealing process used in the production of crystalline structures. In this process, a substance is first melted and then slowly cooled down, forming a crystal. Crystals are arrangements of molecules that result in the minimum potential energy for a system. A system in thermal equilibrium has an energy probability profile that follows a Boltzmann distribution,

$$prob(E,T) = exp(-\frac{E}{kT}) \tag{1}$$

where k is the Boltzmann constant. If the system is taken abruptly out of a temperature T_1 to a lower temperature T_2 before it reaches this equilibrium, there is a chance that high energy sub-states will be present. Thus if a molten substance is *quenched* (cooled down quickly), the system would form a polycrystalline structure – an amorphous state that has a somewhat higher energy. Annealing permits the system to better maintain equilibrium across changing temperatures, allowing for a more reliable arrangement of the substance's particles during cooling.

The SA technique is based in a computer model created by Metropolis et al. [8] to study the properties of a substance at a fixed temperature. The Metropolis algorithm was modified to model changes in temperature as the system cools down [6, 7].

To use the simulated annealing algorithm, a problem is presented in the form of an objective function, together with a feasible (i.e., legal but not necessarily optimal) solution. *Permutations* (small, incremental changes) are applied to this solution, hopefully improving it, until a stable solution is reached. Changes to the arrangement are randomly generated. Decrease in the objective function is considered an improvement; increase is degradation. All improvements ($\Delta E < 0$) are immediately accepted, where ΔE is the difference of energy between the current and the new state ($\Delta E = E_+ - E$). To prevent the algorithm from settling on a local minimum too soon, degradations are sometimes accepted, with a probability $prob(\Delta E, T)$. This probability decreases with decreasing temperature – i.e., higher risk can be taken in earlier stages of the process. Figure 2 summarizes the general algorithm used for this study.

1. Select an initial combination (C_0) and an initial temperature (T_0).
2. Pick the temperature cooling schedule.
3. For each step in the temperature cooling schedule:
   (a) Produce a combination (C_n) by randomly changing the state of the system.
   (b) Calculate the energy differential (ΔE) between the current combination (C) and the new one (C_n)
   (c) Make the new combination (C_n) the current combination (C) if the new combination (C_n):
      - is more efficient ($\Delta E \leq 0$) or
      - satisfies the Metropolis criterion $\xi < prob(\Delta E, T)$ for a random number ξ, $0 \leq \xi \leq 1$ and the annealing temperature T
4. Lower the annealing temperature (T) following the temperature cooling schedule

Figure 2: Algorithm used to perform simulated annealing (SA).

3 Initial Data Allocation using Simulated Annealing

We use the simulated annealing technique to guide the placement of data in a server. Our goal is to generate a data layout that improves the execution of a group of operations. Depending on the level at which we apply this technique, data can be organized in a structured or unstructured way. It may be applied to the tables of a database, the files of a file system, or even at the disk sector level. We consider any operation that requires the access of data in a computer system. For example, this could be sorting an array in memory, filtering out some lines from a file, or more complex operations like querying a database. The process we follow is based on the SA algorithm shown in Figure 2.

3.1 Graph Formulation

We start by building a graph where nodes represent the data storage and computation elements in the system. In this graph, the arcs represent the transfer of data. Lovett et al.[9] showed that remote data accesses are one of the biggest performance bottlenecks in the execution of server workloads. So performance can be improved by a data layout that reduces the number of inter-node transfers. To find a good arrangement for the data, its placement among the nodes must minimize the number of arcs between the 2 resulting sub-graphs. This may be expanded to a system with more than 2 nodes.

Figure 3.a shows the graphic representation of a sample problem. Computation is rep-

resented by P and data by D. Let us assume that the computations and data need be partitioned into 2 nodes. Let us also assume that based on load balancing or other constraints, it is known that the first 2 computations should be in one node and the next 2 in the other. Figure 3.b shows a partition that minimizes the external arcs. The amounts of data transfer needed for each computation may differ. That information can be incorporated into the graph by assigning a weight to each of the arcs. The data partitioning problem gets more complicated now, because the weights also need be considered in determining the partitioning. For example, an arc that corresponds to the use of 1 MB of data would have a smaller impact than an arc representing the consumption of 1 GB. Figure 3.c shows a partition for a situation where one arc carries a significantly higher weight than the other arcs. In a generalized form, the problem consists of reducing the weighted cost of the arcs, as expressed by the formula:

$$cost = \sum a_{ij} w_{ij} \tag{2}$$

where

$$a_{ij} = \begin{cases} 0 & if\ node_i = node_j \\ 1 & otherwise \end{cases} \tag{3}$$

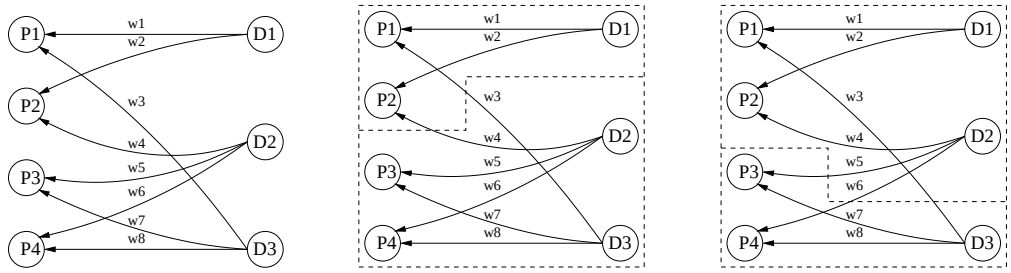

Figure 3: Data transport graph. (a) the initial problem is to arrange the data between 2 nodes. (b) when all weights w_i are identical, the remote transfer cost is $C = w3 + w4$. (c) an arrangement with remote transfer cost $C = w3 + w5 + w6$, when $w4 >> w5 + w6$.

Essentially the problem is a minimum cut (min cut) problem, very commonly employed in several problems including VLSI place and route algorithms. In the example presented above, it is relatively easy to find a solution for the min-cut problem. With only 3 data elements and 2 nodes, there are only 8 possible data layouts.[1] The problem becomes more complicated if we consider more data elements. Using $C(m, r)$ to denote the number of combinations of m chunks of data, placed in r bins, the number of possible layouts for a

[1] The 8 possible layouts are {(D1) (D2,D3)}, {(D2) (D1,D3)}, {(D3) (D1,D2)}, {(D1,D2,D3) ()} and their symmetrical layouts.

2-node configuration is given by the expression,

$$L(m,p) = \sum_{r=p\times m}^{0.5\times m} C(m,r) \tag{4}$$

where p is the minimum percent of the total number of elements that are required to reside in any node. For example, assume there are 40 pieces of data to be arranged in 2 nodes. To balance the data appropriately in the system, the problem might require that no less than 40% of the data be placed in either one of the nodes. Then $m = 40$ and $p = 0.4$, which results in a total of 5.34×10^{11} combinations.

3.2 System State

To adapt the SA technique to the server data placement problem, we need to define the system state. A good representation is a tuple (P, L) that indicates the identity of the data and its location in the system. The state of the system is then the collection of tuples for all the data. Coming up with a new combination as part of the SA algorithm would require to randomly change the location of one of those elements. This approach has two problems:

- If the data elements have very different sizes, then those random changes would have different weights over the solution, therefore affecting the quality of the solution.
- If the data elements are considerably larger than the unit of data transfer, then the allocation might not help reduce the hot-spots in the system.

To solve these problems, whenever possible, we divide the data elements into smaller elements (*chunks*). This is possible for most cases. In a file system, a chunk can be a file or group of files instead of a directory. For databases, we may choose to work with a group of rows instead of the whole table. The number of chunks also determines the length of the allocation processes. Later in the paper, we explore the tradeoff between the size of the chunks and the time it takes to run the algorithm and its effectiveness.

The next task is to define the initial state of the system. To accomplish this, we allocate the data chunks sequentially across the storage elements in the system.

3.3 Objective Function

The goal of the data placement technique could be to improve the execution time, throughput, scalability, fault tolerance, or even power consumption of the server. The objective function must be selected to match this goal. In this paper we use inter-node data traffic and execution time as objective functions.

An objective function expressed in units of time can provide a close match to the problem. Cascaval et al. [10] have shown good results when estimating the time of scientific applications using a model of the form:

$$t_{total} = t_{cpu} + t_{mem} + t_{comm} + t_{i/o} \quad (5)$$

The CPU time is obtained using the number of instructions and estimated hardware costs. The memory time is obtained using a *stack distance model* [11]. The communication and I/O times are obtained from the amount of data that is transferred from the I/O devices or across the nodes. Equation 5 is sufficiently accurate for the purpose of this study.

3.4 Workload

To generate the initial data layout we also need a model of the operations. Commercial workloads offer an advantage due to their modular structure and repetitive behavior. After looking at the high level description of an operation (e.g. a filesystem command, a database query), it is possible to determine the data elements that will be accessed and estimate the cost involved. Our technique uses knowledge about the type of tasks commonly performed by the system to produce an effective data layout. Such a scenario is common in commercial servers. As a result, it is allowed in server benchmarks like TPC-R [3] or SPECweb99 [12].

3.5 Temperature Schedule

The temperature schedule determines the degree of movement permitted during the search. It is selected considering the desired execution time of the algorithm and the desired quality of the final solution. A suitable annealing schedule has an initial temperature (T_0) that is high enough to melt the system completely. The temperature schedule also has a final temperature (T_f) that can freeze the system. This means that while at temperature T_0, the probability of accepting degradations should be close to 1; while at temperature T_f, the probability should be 0. Within this two temperatures is a sequence of monotonically decreasing temperatures. Randelman et al.[13] compared two common schedules: the *exponential cooling scheme* (ECS) and the *linear cooling scheme* (LCS). They found both to be comparable.

In this paper, we use a linear temperature cooling schedule with an initial temperature picked to provide an uphill acceptance probability (χ_0) of 0.8 [14]. Solving Equation 1 for T, we obtain:

$$T_0 = -\frac{C_0}{ln(\chi_0)} \quad (6)$$

where C_0 represents the initial cost of the system. Figure 4 shows the linear temperature

cooling schedule used for one of the experiments. Note that 50 steps are employed.

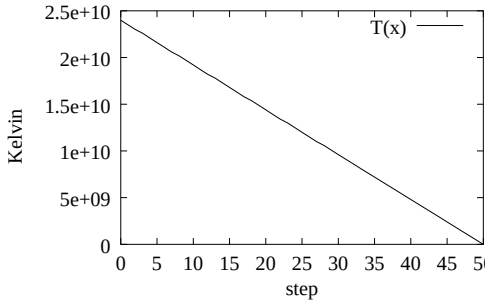

Figure 4: Linear cooling schedule.

Once the cooling schedule is formulated, we start introducing changes to the system. The changes consist in moving a random chunk of data from one node to another. In more complex configurations we would also have to pick a random destination. Figure 5 shows the uphill acceptance probability for each of the steps of the process.

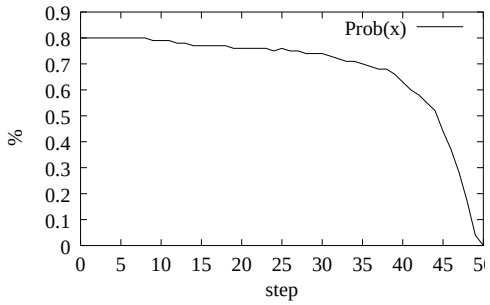

Figure 5: Uphill acceptance probability.

Figure 6 shows the energy of the system for an example experiment. Early in the search, the algorithm is allowed to accept some combinations that increase the value of the objective function. As the algorithm advances, combinations are mostly accepted when they reduce the energy of the system. Figure 6 also shows the energy obtained if we would only accept those solutions that reduce the energy (*Iterative Improvement*). We see that this technique is more likely to settle on a local minimum.

4 Dynamic Reorganization

So far we have presented a technique to organize the data in a system before the operations are executed. We expect this technique to help reduce the time it takes to perform some operations. There are other operations that might not benefit from this consideration. For these, adapting the system at run-time can significantly reduce the time required.

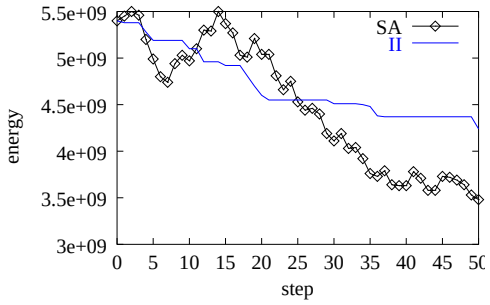

Figure 6: Energy function for the simulated annealing (SA) and the Iterative Improvement (II) algorithms.

Another factor to consider is that commercial server workloads change continuously. In an environment such as an on-line store, we expect the behavior of the shoppers to vary during the day. Early in the day, users start browsing the items and comparing prices. As the day progresses the number of orders might increase. The type of operations changes from mostly reading large volumes of data to modifying a few pieces of data. A different type of change occurs as the popularity of certain items changes. This change has a long term impact and may move the hot-spots in the data from one node to another.

4.1 Description

Evaluating the possible data layouts in real time can be a very expensive task. Our run-time data reorganization technique uses simulated annealing as it allows us to quickly obtain an adequate layout. We periodically assess the state of the data organization and look for possible layouts that would reduce the execution time. At run-time we perform a quick simulated annealing search using a sample of previous and pending operations as the training set. Since this optimization is going to be competing for computation resources with the other operations we may need to tradeoff run time versus quality of the solution. To prevent an inadequate solution from affecting the performance of the system, we only reorganize the data if the solution is capable of significantly improving the performance of the system.

4.2 Reorganization Interval

The selection of the interval between the different adaptations can affect the performance of the system. Determining when to reorganize the data can be done at the boundaries of the operations, groups of operations, at predefined time intervals, or when the number of a certain hardware events (CPU idle cycles, remote memory accesses) passes a certain threshold.

4.3 Temperature Schedule

The temperature schedule is decided in the following way:

- If the number of remote data accesses is high, the initial temperature is also high. A high number of remote data accesses is an indication that the data layout might not be adequate for the workload. A higher temperature allows us to find a better configuration.

- If the number of concurrent operations is high, the initial temperature is low. This decision reduces the impact of the variability of multiple operations.

- If the interconnect utilization is high, the initial temperature is low. Since a reorganization is going to require the use of the interconnect, the technique should not interfere with existing traffic. Alternatively, we can set the size of the data chunks to smaller values. This will make the system adapt over a longer period of time. Also, smaller chunks are less intrusive over the existing data traffic.

Equation 7 shows the expression used to estimate the value of the initial temperature. The value of k_T is selected to maximize the temperature range under optimal conditions.

$$T_0 = k_T T_0' \frac{RA}{OP \times IU} \tag{7}$$

To reduce the overhead of the reorganization, we use a smaller number of steps. This number is smaller than the steps used in a static data allocation.

4.4 Look-ahead Block Selection

At run-time we have information about the operations that have been performed in the past and the data they accessed. Given the nature of the workload, we may also have information about pending operations. It is then possible to estimate which chunks of data are most likely to be used. We use this estimate during simulated annealing to compute the new combination (C_n). Instead of randomly picking a chunk from the collection of all data chunks, we select a data chunk only from the subset of chunks that are likely to be used. This *look-ahead block selection* is useful as it reduces the dimensions of the combination space, which allows a quicker search for an effective arrangement.

5 Evaluation Methodology

We test the effectiveness of both data placement techniques using a decision support system (DSS) workload. Systems designed to run this type of workload are normally implemented as multiprocessor or multicomputer systems. In this section we also discuss the architecture for such a system.

5.1 Computing Platform

Our experiments simulate a distributed shared memory (DSM) server with a system architecture similar to the AlphaServer GS320 [15]. Figure 7 shows a sample topology for a DSM system with 2 nodes, 4 processors each. A DSM server has the advantage of providing a single address space. A single address space allows applications that are designed for a uniprocessor or symmetric multiprocessor system (SMP) to run without modifications. Our experiments are based on a tuned PowerPC based system running AIX 4.3. The database system used is IBM DB2 version 6.1. We conduct these experiments using a full system simulator (SimOS ported to the PowerPC ISA [16]).

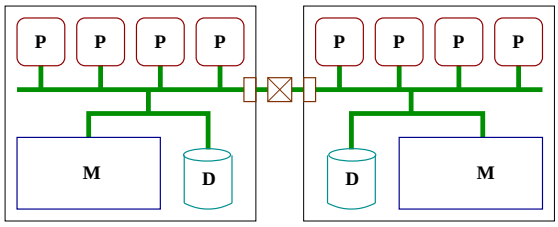

Figure 7: A sample DSM server system. The resources of the system are split among different physical nodes. The system has a global address space, which permits all processors to access data stored in any location. The cost of the access depends on the location of the data.

The DSM system we use has eight identical nodes; each is a 4-way SMP node with 512 MB of memory and 7 disks. Table 1 shows the configuration of each of the nodes. The system has a total of 4 GB of memory and 56 disks. The latency across the network is modeled as 100 ns [17].

Table 1: Configuration of the nodes.

Processors	4 CPUs, 1 GHz
L1-I L1-D L2	128 KB, 64 B block, 2-way 128 KB, 64 B block, 2-way 4 MB, 128 B block, 4-way
Memory	512 MB, 100 ns, 4 banks
System bus	128 bits, 200 MHz, pipelined, split transaction
I/O bus	64 bits, 66 MHz, PCI
Disk bus Disk I/O controller Disk units	160 MB/s, Ultra160 SCSI 2 9.1 GB, 3 ms latency, 7 disks

5.2 Workload

To test the feasibility of the simulated annealing techniques described in the previous sections, we use a group of queries taken from the TPC-R [3] benchmark. The TPC-R benchmark is very similar to the TPC-H [4] benchmark. Both model decision support systems (DSS). While TPC-H models an ad-hoc decision support system, TPC-R models a business reporting system. This allows systems to implement optimizations based on a previous knowledge of the queries.

Table 2 shows a list of the queries used in our experiments. They are picked to include operations with different levels of complexity.

Table 2: Description of selected TPC-R like queries.

Query	Name	Data set size	Implementation
Q1	Pricing Summary Report	1.1 GB	A sequential scan of table *LineItem*. It generates a large number of aggregate values.
Q3	Shipping Priority	2.8 GB	A merge join of tables *Customer* and *Order* and a subsequent merge join of the result with table *LineItem*.
Q6	Forecasting Revenue Change	585 MB	An indexed scan of table *LineItem*.
Q14	Promotion Effect	686 MB	An indexed scan of table *LineItem* and a subsequent merge join with table *Part*.
Q19	Discounted Revenue	902 MB	A merge join of tables *Part* and *LineItem*.

Each of these queries is composed by a group of more simpler operations. The dependencies of these operations are commonly referred as the *query execution plan*. Figure 8.a shows the QEP for our implementation of query 19. Figure 8.b shows the data transfer graph. The weight of the nodes are not shown to provide a better presentation. They are computed statically based on estimates of the amount of data that is going to be returned from each operation. Databases usually provide software performance monitors to track this information. The dotted line indicates the boundaries of the node. The data elements shown inside of of a node correspond to intermediate results. Although accessible by any processor in the system, they are usually accessed by the producer. Therefore, they are not candidates for the data placement.

6 Results

Here we present the results of the simulated annealing method applied to the tasks of static data placement and dynamic data reorganization. We then present the factors that

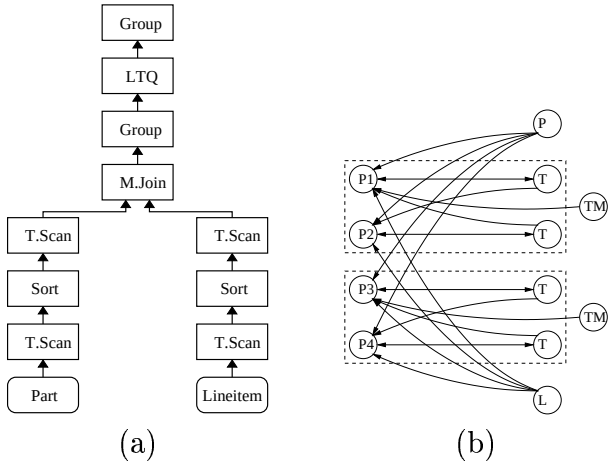

Figure 8: (a) Execution plan for implementation of query 19 of the TPC-R benchmark. (b) Resulting data transfer graph.

affect the effectiveness of the technique.

6.1 Performance

To study how successful the SA technique is in providing an efficient initial data layout, we compare the speedup against a layout designed for maximum data parallelism. This base layout spreads the data for each table equally among the data disks of the nodes using chunks of 1 MB. The SA optimized layout uses 50 steps and data chunks of 16 MB and uses the technique described in Section 3. The objective function is the sum of the inter-node data traffic for the 5 queries. We call this method *global training*, as it uses all the queries to produce a single layout for the system. On the other hand, *local training* produces a using a single query. Using those layouts, we perform detailed simulation of 2 queries of each type. The first is used to warm up the system. We perform measurements on the second query. Figure 9 shows the speedups of both SA layouts over the base layout.

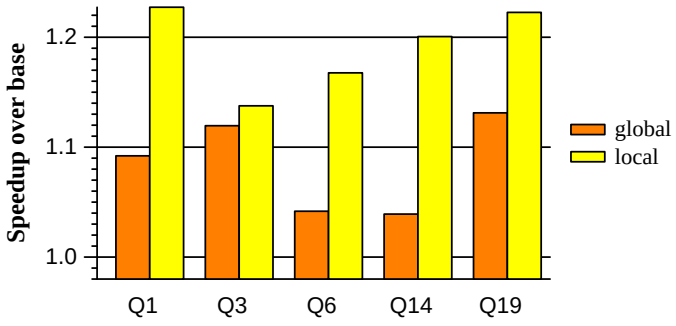

Figure 9: Performance of the simulated annealing static data placement technique.

We observe that SA produces layouts that effectively reduce the execution time of all

the queries. The more practical global training produces speedups of up to 13%. The performance of all queries improve when using local training, where we obtain speedups of up to 22%.

Unfortunately using local training is not always possible. Our alternative solution is to use global training to generate a layout for the system, paired with a SA run-time data reorganization to dynamically adapt the layout. Figure 10 shows the speedups obtained from this scheme. The first bar shows the speedups obtained with the static SA data placement

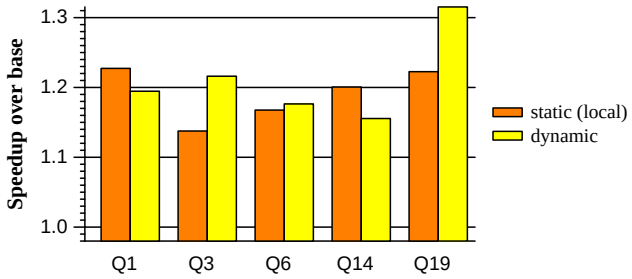

Figure 10: Performance of the simulated annealing run-time data reorganization technique.

technique using local training. The second bar shows the speedups of a static SA layout using global training and the run-time SA data reorganization technique. We observe that the speedups of the dynamic technique approach the speedups of local training static layout. It is interesting to note that it actually produces better results for queries Q3 and Q19, the 2 longest running queries. This is due to the fact that the static layout was produced based on the whole query. Given the length of the queries, the run-time optimizer can adapt the data for the different phases of the queries.

6.2 Sensitivity Analysis

After presenting the basic performance of the SA data placement techniques, we present an analysis of the factors that affect their performance.

6.2.1 Static Data Placement

Steps. The number of steps in the temperature cooling schedule greatly determines the quality of the resulting layout. The time it takes to generate each of these layouts is proportional to the number of steps used. The think time for a 50 step SA takes about 0.87 seconds of user time. This experiment shows the performance for 10, 20, 50 and 70 steps.

Figure 11 shows the results of this experiment. We observe that increasing the number of steps improves the effectiveness of the layout. We do not see a significant improvement between 50 and 70 steps.

Chunk size. We also test the size of the chunks moved for each of the combinations. Using 50 steps and a process similar to the one described above, we change the size of the

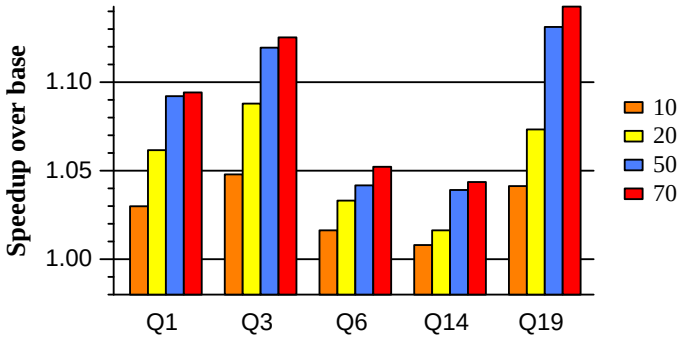

Figure 11: Impact of the number of steps over the simulated annealing process in the static data placement.

chunks from 4 MB to 32 MB. Changing the size of the chunks in our technique affects the number of elements that are part of the optimization process. The more elements we have, the longer it takes to achieve a low energy configuration. Having large elements, on the other hand, reduces the flexibility of the layout. Figure 12 shows our results. We observe that all queries benefit from an intermediate chunk size. The appropriate chunk size needs to be determined for each problem. Empirically we found that an adequate estimate is given by:

$$k \times Steps \times Chunk\ Size \sim Data\ Size \tag{8}$$

for a value $1.5 < k < 2$. We also observe that join queries (Q3, Q14 and Q19) can benefit from smaller chunks.

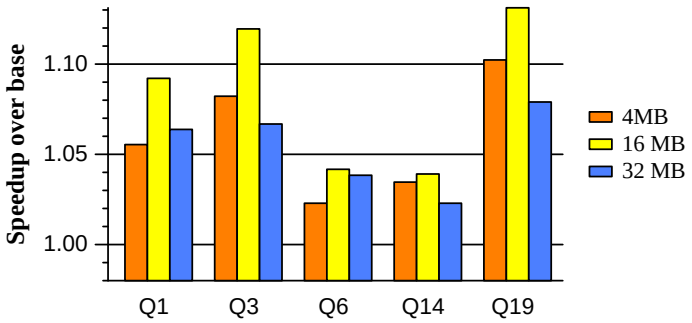

Figure 12: Impact of the chunk size over the simulated annealing process in the static data placement.

Objective function. In these experiments we test which objective function is most effective regarding the SA technique. We use 50 steps and a chunk size of 16 MB. We pick functions that are related to the bottlenecks in the workload, so that reducing them will result in improved performance. These are described in Section 3.3. The first is the amount of traffic due to remote accesses (*inter-node*). The second reflects the time needed to execute

the queries (*time*). Figure 13 shows the results of this experiment. We observe that it is beneficial to use a metric that more closely resembles the metric we shall eventually measure. The time object function outperforms the inter-node object function for all the queries. The only drawback is the time required to complete the optimization process. Optimizing the static data layout using the inter-node object function takes 0.87 seconds. When we use the time object function, the system takes 2.13 seconds to generate the layout. Queries Q6, Q14 and Q19 show a significant improvement when using the time object function, due to the computation intensive nature of the queries. Queries Q1 and Q3 are mostly memory bound, so the inter-node object function is sufficient.

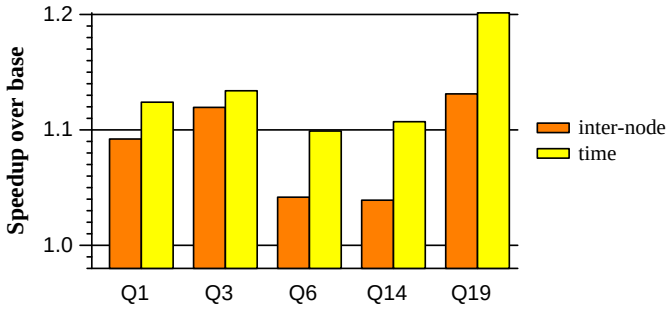

Figure 13: Performance of different objective functions in the simulated annealing process in the static data placement.

6.2.2 Dynamic Data Reorganization

We start with a data layout generated with the static technique using the time objective function, 50 steps, data chunks of 16 MB and global training. Our base model is a system with this data layout and no dynamic data reorganization. Since the dynamic reorganization might require some time to produce a layout adequate for the queries it runs, we experiment with 3 queries of the same type. The first is used to warm up the system. We average the execution times of the second and third queries. Once we start to execute the queries, the system reorganizes the data when it experiences a trigger event. In each invocation, the SA algorithm performs only 10 steps, which execute in approximately 0.082 seconds in the targeted architecture. We note that this duration is not comparable to the times reported in the static data placement. There we use all the queries as the workload to drive the optimization process. Here we use a smaller group of operations, which includes the last 5 operations and (when available) the next 5 operations.

Trigger. In this set of experiments, we study the importance of selecting the right moment to reorganize the data. In the first experiment we invoke this optimization just after completing a query sub-operation. A sub-operation is any operation described as part of the

execution plan (e.g. Q19's execution plan is shown in Figure 8.a). In the second experiment, we only optimize the layout when the number of remote accesses in a given node exceeds 30% of the total accesses. The counter is reset after the optimization is performed. The third experiment optimizes the layout every 0.2 seconds. Figure 14 shows that each of the investigated options is beneficial for a different query. A time based trigger does best on queries Q3 and Q19, which are the longest running queries. An operation based trigger does well for queries Q1 and Q14, which are shorter. Query Q6 benefits from a reorganization based on the fraction of remote accesses. The reason for this is our use of the previous 5 operations and the next 5 operations as a training set. The number of sub-operations in query Q6 is small when compared with the rest of the queries, and its early sub-operations have different data requirements. So each reorganization creates a layout that is not as effective for the remaining sub-operations. For this query, reorganizing the data is better during the early stages, which happen to have the largest number of remote memory accesses.

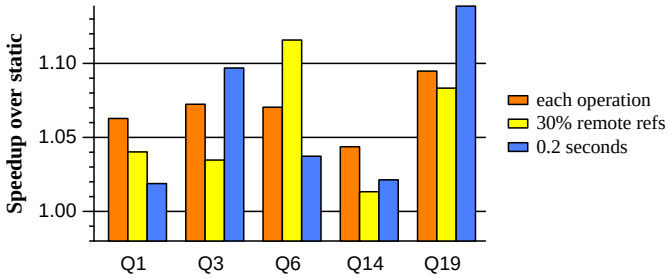

Figure 14: Impact of the trigger event over the simulated annealing process in the dynamic data reorganization.

Steps. Figure 15 shows the effect that the number of steps has on the execution time. The number of steps has an impact on the quality of the layout and hence in the execution time. In this experiment we show that the length of the optimization also plays a role on performance. As mentioned earlier, performing 10 steps takes approximately 0.082 seconds. For this experiment we also use 5 and 15 steps, which take approximately 0.051 seconds and 0.123 seconds.

Objective Function. We observed earlier that using time as the objective function generates a better layout for the static data placement. In the run-time data reorganization we have the additional constraint that any optimization has to be performed quickly. Figure 16 shows that the additional time needed to use that objective function results in a smaller return than the use of the inter-node data transfer objective function.

Select. In the previous experiments we observe that the duration of the optimization stage can impose a toll on the result. This can be because the system does not have enough time to find a more effective layout, or because by the time the system finds it, the situation

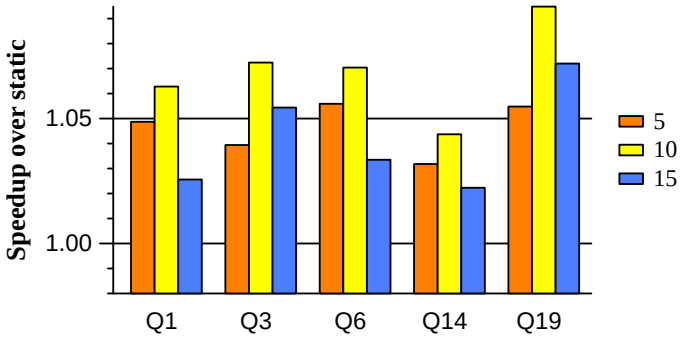

Figure 15: Impact of the number of steps over the simulated annealing process in the dynamic data reorganization.

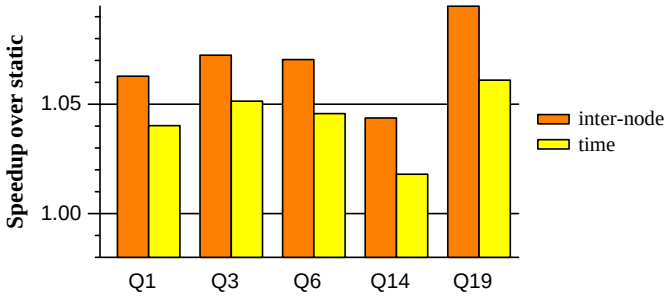

Figure 16: Impact of the objective function over the simulated annealing process in the dynamic data reorganization.

has already changed. To handle this, we can make use of the locality of data references. In these experiments we test the look-ahead block select described in Section 4.4. We compare it with the random block select. Figure 17 shows the speedups obtained by both selection policies.

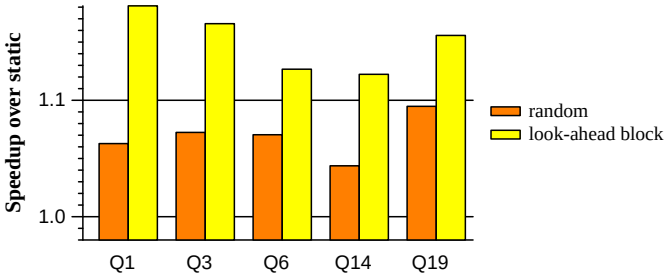

Figure 17: Impact of the combination selection heuristic over the simulated annealing process in the dynamic data reorganization.

7 Conclusions

Commercial applications access large volumes of data, which spread over multiple storage locations in modern servers. A poor layout of the data can hurt the performance and

scalability of the system.

We propose using the Simulated Annealing technique to efficiently place the data in a server and to reorganize the data to adapt to the needs of the workload. Our work focuses on databases, but it can be applied to other large scale systems like web servers and file servers.

We use a DSS benchmark to test the initial data placement and find speedups of up to 13% when using a global training set. We observe speedups of up to 19% with the run-time data reorganization technique when starting with an optimized layout. Starting from the globally optimized data layout, dynamic data reorganization using simulated annealing produces speedups of up to 31%. This is due to the ability of the technique to tune the layout to the different phases of the query. The speedups shown approximate those seen with a local training mode, which shows the merit of the SA technique in providing an efficient run-time reorganization.

References

[1] D. M. Schuler and E. G. Ulrich, "Clustering and linear placement," in *Proceedings of the 9th ACM/IEEE Design Automation Conference*, pp. 50–56, 1972.

[2] K. Shahookar and P. Mazumder, "VLSI cell placement techniques," *ACM Computing Surveys (CSUR)*, vol. 23, pp. 143–220, June 1991.

[3] "TPC-R specification."
http://www.tpc.org/tpcr.

[4] "TPC-H specification."
http://www.tpc.org/tpch.

[5] F. Gavril, "Some NP-complete problems on graphs," in *Proceedings of the 11th Conference on Information Sciences and Systems*, pp. 91–95, 1977.

[6] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 13 1983.

[7] E. H. L. Aarts and P. J. M. van Laarhoven, "Statistical cooling: A general approach to combinatorial optimization problems," *Philips Journal of Research*, vol. 40, pp. 193–226, 1985.

[8] N. C. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[9] T. Lovett and R. Clapp, "STiNG: A CC-NUMA computer system for the commercial marketplace," in *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, (Philadelphia, PA, USA), pp. 308–317, May 22–24 1996.

[10] C. Cascaval, L. D. Rose, D. A. Padua, and D. A. Reed, "Compile-time based performance prediction," in *Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing*, (La Joya, CA, USA), pp. 365–379, Aug. 1999.

[11] C. Cascaval, "Estimating cache misses and locality using stack distances," in *Proceedings of the 17th Annual ACM International Conference on Supercomputing*, (San Francisco, CA, USA), June 23–26 2003.

[12] "SPECweb99 benchmark."
http://www.specbench.org/osg/web99.

[13] R. Randelman and G. Grest, "N-city traveling salesman problem - optimization by simulated annealings," *Journal of Statistical Physics*, vol. 45, pp. 885–890, 1986.

[14] S. Kirkpatrick, "Optimization by simulated annealing - quantitative studies," *Journal of Statistical Physics*, vol. 34, pp. 975–986, 1984.

[15] K. Gharachorloo, M. Sharma, S. Steely, and S. V. Doren, "Architecture and design of AlphaServer GS320," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, (Boston, MA, USA), pp. 13–24, Nov. 13–15 2000.

[16] "SimOS PowerPC."
http://www.research.ibm.com/arl/projects/SimOSppc.html.

[17] M. M. Michael, A. K. Nanda, B.-H. Lim, and M. L. Scott, "Coherence controller architectures for SMP-based CC-NUMA multiprocessors," in *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-97)*, (Denver, CO, USA), pp. 133–143, June 2–4 1997.