Improving Server Performance on Transaction Processing Workloads by Enhanced Data Placement

Juan Rubio
Lab. for Computer Architecture
The University of Texas at Austin
Austin, TX 78712
jrubio@ece.utexas.edu

Charles Lefurgy
IBM Austin Research Lab
Austin, TX 78758
lefurgy@us.ibm.com

Lizy K. John
Lab. for Computer Architecture
The University of Texas at Austin
Austin, TX 78712
ljohn@ece.utexas.edu

Abstract

Modern servers access large volumes of data while running commercial workloads. The data is typically spread among several storage devices (e.g. disks). Carefully placing the data across the storage devices can minimize costly remote accesses and improve performance.

We propose the use of simulated annealing to arrive at an effective layout of data on disk. The proposed technique considers the configuration of the system and the cost of data movement. An initial layout globally optimized across all queries, shows speedups of up to 13% for a group of DSS queries and up to 6% for selected OLTP queries.

This technique can be re-applied at run-time to further improve performance beyond the initial, globally optimized data layout. This scheme monitors architecture parameters to prevent optimizations of multiple operations to conflict with each other. Such a dynamic reorganization results in speedups of up to 23% for the DSS queries and up to 10% for the OLTP queries.

1. Introduction

Modern servers access large volumes of data while executing emerging commercial workloads. Servers typically have multiple processing elements, whether clusters of computers or tightly coupled multiprocessors. The storage devices, which may be located in different nodes within the system, are not all equally accessible by the processing elements. Local accesses are usually 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. Commercial workloads have a large data footprint, and operations commonly require data that is not memory resident. Hence, one important factor to consider is the placement of data on disk; an efficient placement minimizes remote data transfers at run-time.

This paper formulates data placement as a combinatorial optimization problem and uses simulated annealing to arrive at a good solution. Combinatorial optimizations are used in many branches of sciences to find the best arrangement of objects for a given constraint. For example, they are used in the context of VLSI and Electronic Design Automation to solve placement and route problems [18, 19].

The paper is organized as follows. Section 2 introduces the simulated annealing technique. Section 3 explains how the data placement problem can be formulated as a graph and how simulated annealing can be applied to find a good initial placement. Section 4 presents the run-time data reorganization technique. Section 5 describes our performance evaluation methodology. Section 6 analyzes the results of the proposed technique. Section 7 compares our approach to previous work. Section 8 presents our conclusions.

2. Simulated Annealing

This paper addresses the placement of data across the nodes of a server. Our main goal is to reduce the number of remote data accesses. Selecting the optimal placement has been shown to be NP-complete for general graphs [6], so most systems use heuristics to approximate a solution.

Simulated Annealing (SA) [12, 1] is an algorithm that quickly and effectively optimizes a solution over a large state space. It does not guarantee the optimal solution, but can produce a solution close to the global minimum in much less time than an exhaustive search. SA starts with a configuration and searches for better solutions by making random modifications (permutations). Permutations resulting in improved performance are applied to the system. However, to escape local minima, the SA algorithm accepts some solutions that do not perform well. The probability of accepting a harmful change decreases as the algorithm progresses.

The acceptance probability is determined by the cost of the change and the *temperature* T, an artificial parameter representing the algorithm's progress.

$$prob(\Delta C, T) = exp(-\frac{\Delta C}{kT}) \quad (1)$$

1. Set the initial solution; $S = S_0$
2. Set the initial temperature; $T = T_0$
3. Repeat while $T > T_f$:
   (a) For each permutation at this temperature step:
      i. Produce a new solution S_{new} by a random permutation of S
      ii. Calculate the cost differential ΔC between S and S_{new}
      iii. Set S to S_{new} if:
         - $\Delta C \leq 0$ or
         - $\xi < prob(\Delta C, T)$ for a random number ξ, $0 \leq \xi \leq 1$
   (b) Move to the next temperature step by reducing T according to the cooling schedule

Figure 1. Simulated annealing algorithm.

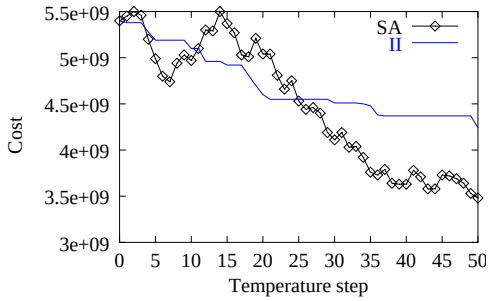

Figure 2. Cost function for the simulated annealing (SA) and the Iterative Improvement (II) algorithms.

The algorithm starts at a high temperature, T_0, and reduces the temperature at each iteration. The first iteration begins with a feasible (i.e., legal but not necessarily optimal) solution with cost C_0. When the temperature reaches zero, the probability of accepting worse solutions is zero and the algorithm halts. Figure 1 summarizes the general algorithm used for this study.

Figure 2 shows the cost obtained by only accepting solutions that reduce the energy (*Iterative Improvement* – II) versus the SA algorithm. The II technique is more likely to settle on a local minimum.

When applying SA to data placement, C becomes the effectiveness of using a particular data placement. Example functions are the amount of inter-node data traffic or the time to complete the workload. The values for k and T_0 are discussed in sections 3.3 and 4.2. The reduction of temperature on each iteration is determined by the temperature cooling schedule, which is described in Section 5.

3. Static Data Placement

In this section, we evaluate the benefit of performing an initial data placement before the workload is executed.

3.1. Solution State

The solution state used in the SA algorithm describes the location of data in the server system. A data element could be a file or a database table. Whenever possible, we divide the data elements into smaller, fixed-sized *chunks*, which allows a more balance distribution of data across the disk of the system. For our workloads, a chunk consists of multiple rows in a database table. Each chunk is represented by a pair (D, L), which indicates the identity the block of data and its location in the system. A solution state is a list of all the tuples in the server system.

The initial solution state of the system is found by allocating the chunks sequentially across the storage elements in the system. The solution state is permuted by selecting a chunk and moving it to a different node in the server.

3.2. 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 SA objective function must be selected to match this goal. In this paper we use inter-node data traffic and execution time as objective functions.

3.2.1. Inter-node Data Traffic Lovett et al. [13] showed that remote data accesses are one of the biggest performance bottlenecks in the execution of server workloads. Performance can be improved by a data layout that reduces the number of inter-node transfers. Therefore, we consider using inter-node data traffic as an objective function.

Data traffic in the system can be represented by a graph. The nodes in the graph represent the data storage and computation elements in the system, and arcs represent the transfer of data. Figure 3 shows the graphic representation of a sample problem. Computation is represented by P and data by D. Dotted lines represent node boundaries. 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 objective function is the sum of the weight of all arcs which cross the boundaries of a node, as expressed by the formula:

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

where

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

3.2.2. Time We also consider an objective function expressed in units of time to run the workload. This function yields higher quality solutions, but the algorithm has a longer running time than the objective function based on remote data accesses. Cascaval et al. [4] have shown good

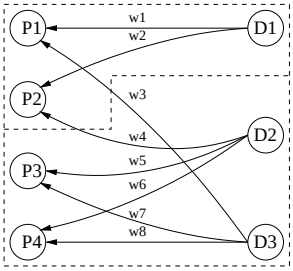

Figure 3. Data partition graph. The data is arranged between 2 nodes, resulting in a remote transfer cost $C = w3 + w4$.

results when estimating the time of scientific applications using a model of the form:

$$t_{total} = t_{cpu} + t_{mem} + t_{comm} + t_{i/o} \quad (4)$$

The CPU time is obtained using the number of instructions and estimated hardware costs. The memory time is obtained using a *stack distance model* [3]. The communication and I/O times are obtained from the amount of data that is transferred across the system's nodes or from the I/O devices. Equation 4 does not consider the overlap between these components, but it is sufficiently accurate for the purpose of this study.

3.3. Temperature Schedule for Static Placement

For all experiments, we reduce the temperature linearly. We use 50 temperature steps for the initial data placement. For each temperature step, 20 permutations to the solution are evaluated. Since our application of simulated annealing does not correspond to any real physical phenomenon, we can substitute $k = 1$ in Equation 1. We find the value for T_0 by using an initial uphill acceptance probability of 0.8 [11] and solving Equation 1 for T.

$$T_0 = -\frac{C_0}{ln(0.8)} \quad (5)$$

4. Dynamic Data Reorganization

Different stages of a workload can use data differently. For example, in a commercial environment such as an on-line store, user behavior varies during the day [15]. The changing popularity of store merchandise can have a longer term impact, moving the hot-spots in the data from one node to another. In this type of situation, system performance can benefit by re-optimizing the data placement during workload execution. This section presents modifications to allow for dynamic data placement.

4.1. Description

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 queries as the training set.

Since dynamic reorganization competes for the same resources as the main computation, it can hurt performance. To prevent this, we shorten the search time by reducing the temperature cooling schedule to 10 steps. When the system has a heavy load, selecting a lower initial temperature reduces the number of data chunks transferred to implement the new layout. Finally, data is only reorganized if the new layout is expected to significantly improve performance.

4.2. Temperature Schedule for Dynamic Placement

Equation 6 shows the expression used to estimate the value of the initial temperature T_0'. When the system is at high utilization, the initial temperature is lower to decrease the probability of transfer-intensive solutions. When the system is experiencing a large number of remote data accesses, the initial temperature is higher to encourage SA to find better solutions.

$$T_0' = k_T T_0 \frac{RA}{OP \times IU} \quad (6)$$

The variables are:

- T_0: from equation 5.
- RA (remote accesses): This is the ratio of remote access to all data accesses. A high number of remote data accesses is an indication that the data layout might not be adequate for the workload. In this case, a higher temperature is obtained, allowing greater freedom to find a better configuration.
- OP (operations): This is the number of concurrent operations in the system. If this number is high, then the initial temperature is low. This reduces the impact of the variability of multiple operations.
- IU (interconnect utilization): This is a ratio expressing the utilization of the interconnect. Since a reorganization is going to require the use of the interconnect, a high IU reduces the temperature, preventing interference with existing traffic.
- k_T: This value is selected so that T_0' approaches T_0, i.e., to maximize the temperature, when the system is underutilized. In the experiments, we use $k_T = 18$.

4.3. Look-ahead Block Selection (LABS)

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 new solutions. Instead of randomly picking a data chunk from the

collection of all chunks, we select a chunk only from the subset that are likely to be used. This *look-ahead block selection* is useful as it reduces the search space, shortening the search for an effective arrangement.

4.4. Simulated Annealing Migration (SAM)

Some workloads are characterized by short operations, such as in on-line transaction processing (OLTP). For these workloads, the cost of data reorganization is relatively high compared to the cost of performing an operation. This severely limits the frequency at which reorganizations could be performed. In this case, we adapt the data reorganization technique to take advantage of the remote data accesses of ongoing operations. Instead of transferring data chunks based on LABS, the SA algorithm decides if a chunk of data being accessed remotely should be made local to the accessing node. If so, the chunk is deallocated from its initial location and stored in a disk belonging to the new node.

5. Evaluation Methodology

In this section, we discuss the simulation model, the workloads, and the computation of the cost function.

5.1. Computing Platform

We simulate a DSM server with a system architecture similar to the AlphaServer GS320 [7]. The simulated DSM server has 4 clusters, each with 4 processors for a total of 16 processors. A DSM server has the advantage of providing a single, global address space. A single address space allows applications that are designed for a uniprocessor or symmetric multiprocessor system (SMP) to run without modification. Our experiments are based on a tuned PowerPC system running AIX 4.3. The database system is IBM DB2 version 6.1. We conduct these experiments using a full system simulator (SimOS ported to the PowerPC ISA [8]).

The DSM system has four 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 2 GB of memory and 28 disks. The latency across the network is 100 ns [16].

5.2. DSS Workload

We use a group of decision support system (DSS) queries similar to the TPC-H [21] benchmark. The database schema is similar to the schema specified by the benchmark and the tables are populated using a scale factor of 1. Table 2 shows a list of the queries used in our experiments. They are picked to include operations with different levels of complexity. Sequential scan, indexed scan, and merge join operations are included. The tables vary in size from Region (1 KB), the smallest, to LineItem (813 MB), the largest.

Processors	4 CPUs, 1 GHz, assume 1 IPC
L1-I	128 KB, 64 B block, 2-way
L1-D	128 KB, 64 B block, 2-way
L2	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	160 MB/s, Ultra160 SCSI
Disk I/O controller	2
Disk units	9.1 GB, 3 ms latency, 7 disks

Table 1. Configuration of the nodes.

5.3. OLTP Workload

We also use a group of queries produced by 4 web interactions based on the TPC-W benchmark [22]. To generate the workload we use a real system – not simulated. The system is populated with 10,000 items and supports 280 emulated browsers. We apply a stream of 10 interactions of the same type to our system and create a trace of the queries performed by the database. We then use this sequence of queries as our workload. Table 3 describes the interactions used in our experiments. The *shopping cart* and *buy confirm* interactions involve reading and modifying data, whereas *best seller* and *product description* only read data. The first two queries provide test cases for dynamically changing data.

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 examine the factors that affect the effectiveness of the technique.

6.1. Performance of DSS queries

The system starts with a database layout designed to exploit intra-partition 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. Our *global training* method uses information on all the queries to produce a single layout for the system; the objective function is the sum of the inter-node data traffic for the 5 queries. Conversely, *local training* produces a layout optimized for a single query. The first two bars of Figure 4 show the speedups of both SA layouts over the base layout.

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 per-

Query	Name	Data set size	Duration	Implementation
Q1	Pricing Summary Report	1.1 GB	1.1511 s	A sequential scan of table *LineItem*. It generates a large number of aggregate values.
Q3	Shipping Priority	2.8 GB	10.6040 s	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	1.3162 s	An indexed scan of table *LineItem*.
Q14	Promotion Effect	686 MB	3.2236 s	An indexed scan of table *LineItem* and a subsequent merge join with table *Part*.
Q19	Discounted Revenue	902 MB	13.5045 s	A merge join of tables *Part* and *LineItem*.

Interaction	Name	Data set size	Duration	Description
I.Shop.Cart	Shopping cart	55.4 MB	0.6955 s	Reads the customer, item and shopping cart tables, and sometimes stores a new item in the shopping cart.
I.Buy.Conf.	Buy confirm	160.6 MB	0.7589 s	Reads the customer, address and shopping cart information. It creates a new order
I.Best.Sell	Best sellers	258.0 MB	0.6324 s	Obtains the top selling items after looking at the mostrecent orders, it also includes the author information for those items.
I.Prod.Desc	Product description	9.7 MB	0.3574 s	Displays the author and details of a given item.

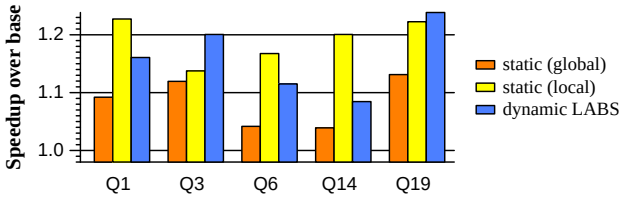

Figure 4. Performance of the SA initial and run-time data reorganization for DSS queries.

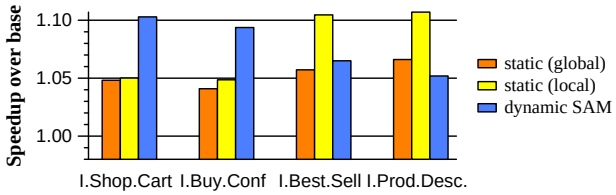

Figure 5. Performance of the SA initial and run-time data reorganization for OLTP queries.

formance of all queries improve further with 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. In this scheme, dynamic reorganization occurs after every operation. The third bar of Figure 4 shows the speedups obtained from this scheme. We observe that the speedups of the dynamic technique approach the speedups of local training static layouts. 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 these queries, the run-time optimization can adapt the data for the different operations of the queries.

6.2. Performance of OLTP queries

Next, we study the use of the SA data placement technique for an OLTP workload. We start with a similar base layout, which exploits intra-partition parallelism. As with the static layouts produced for the DSS workload, we use 50 steps, data chunks of 16 MB, and inter-node data traffic as the cost function for the SA process. The globally-optimized layout is obtained using a training set of queries from the four types of interactions in Table 3. The locally-optimized layout only uses one type of interaction.

Figure 5 shows that the global training static layout only improves the performance of the OLTP interactions by up to 6.6%. Local training is more useful for this workload, producing speedups of up to 10%. However, due to the complexity of the first two interactions (**I.Shop.Cart** and **I.Buy.Conf.**), the performance of the local training layouts does not improve on that achieved with global training. The third interaction (**I.Best.Sell**) performs a join of 2 tables, similar to query **Q19** of the DSS workload; thus it benefits greatly from a static local layout.

The run-time data reorganization improves the performance of most of the interactions, producing speedups of up to 10% over the base layout. The **I.Prod.Desc.** interaction is an exception. Due to its short duration, not enough data reorganizations can be completed to achieve the performance of a static local layout.

6.3. Sensitivity Analysis

In this section we analyze the factors that affect the performance of the SA data placement techniques.

6.3.1. Static Data Placement Steps. The number of steps in the temperature cooling schedule greatly affects the quality

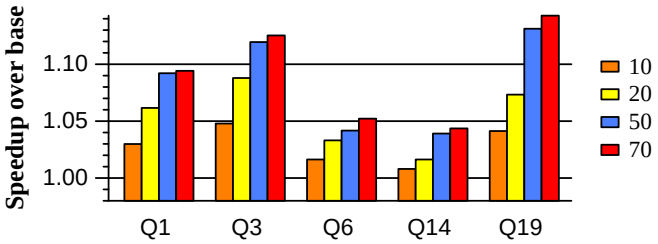

Figure 6. Impact of the number of steps over the SA static data placement.

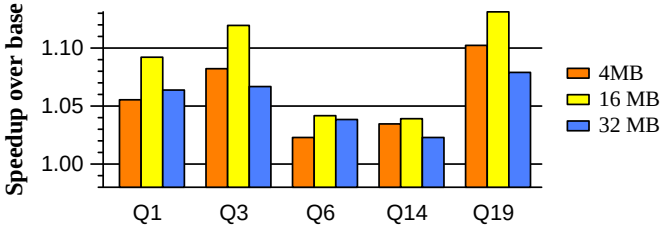

Figure 7. Impact of the chunk size over the SA static data placement.

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 time on one processor. This experiment shows the performance for 10, 20, 50 and 70 steps.

Figure 6 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 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 7 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:

$$k \times Steps \times Chunk\ Size \sim Data\ Size \qquad (7)$$

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

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 3.2. 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 8

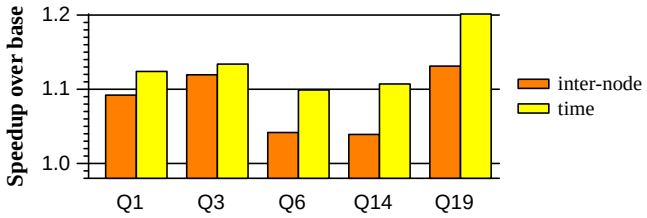

Figure 8. Performance of different objective functions in the SA static data placement.

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 objective function outperforms the inter-node objective 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 objective function takes 0.87 seconds. When we use the time objective function, the system takes 2.13 seconds to generate the layout. Queries Q6, Q14 and Q19 show a significant improvement when using the time objective function, due to the computation intensive nature of the queries. Queries Q1 and Q3 are mostly memory bound, so the inter-node objective function is sufficient.

6.3.2. 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 after every operation. 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.

Steps. Figure 9 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

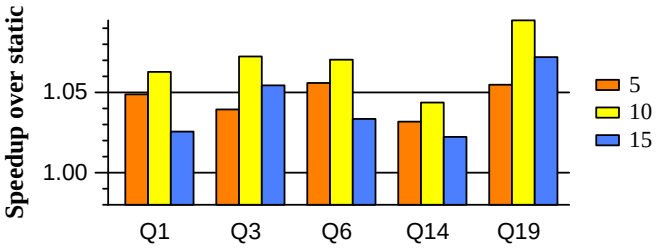

Figure 9. Impact of the number of steps over the SA dynamic data reorganization.

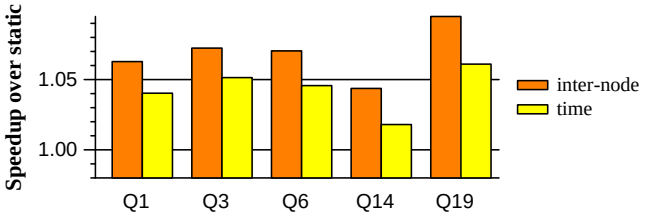

Figure 10. Impact of the objective function over the SA dynamic data reorganization.

performed quickly. Figure 10 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.

Selection Policy. 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 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 4.3. We compare it with the random block select. Figure 11 shows the speedups obtained by both selection policies.

7. Related Work

Combinatorial optimizations, including simulated annealing, have been used in the context of VLSI and Electronic Design Automation to solve placement and route problems [18, 19]. In the field of computer architecture, the uses of combinatorial optimizations have been limited mostly to genetic algorithms used to explore the design space of microprocessor components [20, 5, 2]. Simulated annealing has also been used to cluster data when performing pattern recognition [10], and to produce optimized query plans [9].

Work by Tsangaris et al. looked at stochastic techniques for clustering objects [23]. Additional work shows that randomizing algorithms produce the best clustering, but the cost involved is too high for their application [24].

A study by McErlean et al. shows that simulated annealing can also be used to cluster data in a database [14]. Their work used real runs in a database instead of a cost function. Therefore it required many hours to generate a single layout. Our work shows that picking an adequate cost function can produce a layout in less than a second.

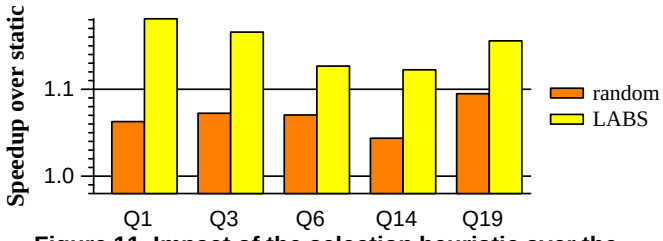

Figure 11. Impact of the selection heuristic over the SA dynamic data reorganization.

Work by Rao et al. uses genetic algorithms to find a good partition for a group of queries in a shared-nothing database cluster [17]. A partition is a description of the logical group of nodes (nodegroup) over which tables and indices are evenly allocated. Our approach is similar to theirs if we think of a single node of our CC-NUMA system as a nodegroup. However, we opt for a finer grained approach that allows small chunks of data to be allocated unevenly across the different disks of the system instead of evenly splitting tables across each nodegroup. Additionally, their work considers partitioning as a part of query optimization. We look at data reorganization as a way to optimize the layout further for an existing set of query plans.

8. Conclusions

Commercial applications access large volumes of data. Because this data is spread over multiple storage locations, operations commonly perform a large amount of remote data accesses. A poor data layout can hurt the performance 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. We demonstrate the feasibility and effectiveness of this approach on DSM servers running DSS and OLTP queries from the TPC-H and TPC-W benchmarks. Our work can also be applied to other workloads, such as web and file serving.

We test the initial data placement and find speedups of up to 13% when using a global training set. Starting from the globally optimized data layout, dynamic data reorganization produces speedups of up to 23%. The speedups shown approximate those seen with the local training mode, which shows the merit of the SA technique in providing an efficient run-time reorganization.

By dynamically tuning the data layout, we can adapt it to the different phases of the workload. When the current phase is performing local accesses, the SA technique uses information about the pending operations to prefetch remote chunks of data that might later be used in another phase. This opera-

tion reduces the access latency of future accesses and the potential for contention in the global interconnect.

Our technique monitors the state of the hardware to decide if reorganization is worth the effort. This aspect of the dynamic reorganization is especially useful in an OLTP scenario, where multiple queries are running at once, greatly complicating the decision of what chunks of data to transfer. The use of hardware information makes this SA technique adaptable and effective for a server running complex operations like the ones seen in commercial workloads.

Acknowledgments

We want to thank Madhavi Valluri, Karthick Rajamani and the anonymous reviewers for their suggestions to help improve the quality of this paper. This research is supported by the Defense Advanced Research Projects Agency under contract F33615-01-C-1892, NSF grant ECS-0113105, an IBM SUR grant, and by Tivoli, Intel and IBM.

References

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

[2] D. H. Albonesi and I. Koren. STATS: A framework for microprocessor and system-level design space exploration. *Journal of Systems Architecture*, 45(12-13):1097–1110, June 1999.

[3] 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.

[4] 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*, pages 365–379, La Joya, CA, USA, Aug. 1999.

[5] J. Emer and N. Gloy. A language for describing predictors and its application to automatic synthesis. In *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-97)*, pages 304–314, Denver, CO, USA, June 2–4 1997.

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

[7] 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*, pages 13–24, Boston, MA, USA, Nov. 13–15 2000.

[8] IBM Corporation. SimOS PowerPC. http://www.research.ibm.com/arl/projects/SimOSppc.html.

[9] Y. E. Ioannidis and E. Wong. Query optimization by simulated annealing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-87)*, pages 9–22, San Francisco, CA, USA, May 27–29 1987.

[10] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, Sept. 1999.

[11] S. Kirkpatrick. Optimization by simulated annealing - quantitative studies. *Journal of Statistical Physics*, 34:975–986, 1984.

[12] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 13 1983.

[13] 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 (ISCA-96)*, pages 308–317, Philadelphia, PA, USA, May 22–24 1996.

[14] F. J. McErlean, D. A. Bell, and S. I. McClean. The use of simulated annealing for clustering data in databases. *Information Systems*, 15(2):233–245, 1990.

[15] D. A. Menasce and V. Akula. Towards workload characterization of auction sites. In *6th Annual IEEE Workshop on Workload Characterization (WWC-6)*, Oct. 2003.

[16] 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)*, pages 133–143, Denver, CO, USA, June 2–4 1997.

[17] J. Rao, C. Zhang, G. Lohman, and N. Megiddo. Automating physical database design in a parallel database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-2002)*, pages 558–569, Madison, WI, USA, June 4–6 2002.

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

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

[20] T. J. Stanley and T. Mudge. Systematic objective-driven computer architecture optimization. In *Proceedings of the 16th Conference on Advanced Research in VLSI*, pages 286–300, Mar. 27–29 1995.

[21] Transaction Processing Council. The TPC-H benchmark specification. http://www.tpc.org/tpch.

[22] Transaction Processing Council. The TPC-W benchmark specification. http://www.tpc.org/wspec.html.

[23] M. M. Tsangaris and J. F. Naughton. A stochastic aproach for clustering in object stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-91)*, pages 12–21, Denver, CO, USA, May 29–31 1991.

[24] M. M. Tsangaris and J. F. Naughton. On the performance of object clustering techniques. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-92)*, pages 144–153, San Diego, CA, USA, June 1992.