

Parallel Database Sort and Join Operations Revisited on Grids

Werner Mach and Erich Schikuta

University of Vienna

Department of Knowledge and Business Engineering
Research Lab Computational Technologies and Applications
Rathausstraße 19/9, A-1010 Vienna, Austria
werner.mach@univie.ac.at, erich.schikuta@univie.ac.at

Abstract. Based on the renowned method of Bitton et al. (see [1]) we develop a concise but comprehensive analytical model for the well-known Binary Merge Sort, Bitonic Sort, Nested-Loop Join and Sort Merge Join algorithm in a Grid Environment.

We concentrate on a limited number of characteristic parameters to keep the analytical model clear and focused. Based on these results the paper proves that by smart enhancement exploiting the specifics of the Grid the performance of the algorithms can be increased and some results of Bitton et al. for a homogenous multi-processor architecture are to be invalidated and reversed.

1 Introduction

Today the Grid gives access to viable and affordable platforms for many high performance applications hereby replacing expensive supercomputer architectures.

Basically the Grid resembles a distributed computing model providing for the selection, sharing, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users quality-of-service requirements (see [2]).

We believe that the Grid delivers a suitable environment for parallel and distributed database systems. There is an urgent need for novel database architectures due to new stimulating application domains, as high energy physics experiments, bioinformatics, drug design, etc., with huge data sets to administer, search and analyze. This situation is reflected by a specific impetus in distributed database research in the Grid, starting with the Datagrid project [3] and now mainly carried by the OGSA-DAI project [4]. After a few years where the research was mainly in the area of distributed data management, now a new stimulus on research on parallel database operators focusing Grid architectures (see [5], [6]) can be noticed.

Sorting and Joining are extremely demanding operation in a database system. They are the most frequently used operators in query execution plans generated

by database query optimizers. Therefore their performance influences dramatically the performance of the overall database system [7], [8]. Generally these algorithms can be divided into internal (main memory based) and external (disk based) algorithms [9]. An external algorithm is necessary, if the data set is too large to fit into main memory. Obviously this is the common case in database systems.

In this paper we present an analysis and evaluation of the most prominent parallel sort and join algorithms, Binary Merge Sort and Bitonic Sort, and Nested-Loop and Sort Merge Join in a Grid architecture based on the well known analysis and findings of Bitton et al. [1]. Now these algorithms are reviewed under the specific characteristics of the Grid environment. The surprising fact, justified by the findings of this paper and resulting from the characteristic situation of the Grid, is that the some results on the general performance of these parallel algorithms are invalidated and reversed, i.e. in a Grid environment the performance of these algorithms can be improved choosing an adapted workflow layout on the Grid taking smartly into account the specific node characteristics.

The paper is organized as follows. In the next section the architectural framework of the Grid environment is laid out. This is followed by the description of the parallel algorithms, their specific characteristics and the definition of the basic parameters of the analysis. In section 5 a comprehensive analytical formulation of the parallel algorithms is given, followed by the evaluation of the algorithms compared and discussion of the findings. The paper is closed by a conclusion and a presentation of topics for further research.

2 Architectural Framework

2.1 A Generalized Multiprocessor Organization

The work of Bitton et al. [1] is based on a so called generalized multiprocessor organisation, which basically comprises a *homogenous* classical supercomputer architecture, where every working node shows the same characteristics in processing power, every disk node in I/O performance and the network interconnect bandwidth is the same for all node. So basically for the evaluation of the parallel operations on the generalized multiprocessor organization only the following components are considered:

1. a set of general-purpose processors,
2. a number of mass storage devices,
3. an interconnect device connecting the processors to the mass storage devices via a high-speed cache

Such an organization is depicted by Figure 1.

2.2 A Static Simplified Grid Organization

The big difference of a Grid environment, which is the focus of this paper, to the architecture laid out above is the *heterogeneity* of all comprising elements,

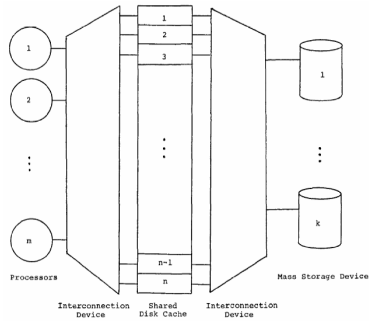


Fig. 1. Generalized Multiprocessor Organization (from [1])

which are processing nodes, interconnect bandwidth, disk performance. For the analytical comparison of the parallel algorithms in focus we restrict our approach to a simplified Grid computing organization focusing on the sensitive parameters of the model in focus.

We use the term *Static Simplified Grid Organization*, which describes an organization to perform a distributed query on a loosely coupled number of heterogeneous nodes. There is no logical order or hierarchy. That means, there is no logical topology of the nodes (e.g. no master/slave nor a equivalent order). Each node has a fixed number of properties with defined values. The term *Static* is used to describe that the values of each node and also the speed of the network are fixed and not changeable during the execution of a query. The sketch of such an organization is shown in Figure 2. These assumptions build the basis for our approach to analyze the operations in a simplified Grid.

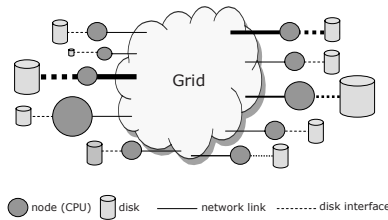


Fig. 2. Static Generalized Grid Organization

2.3 General Architectural Overview

The general layout of our architecture is as follows: A number of nodes are connected via a network. A node consists of one ore more CPUs (Central Processing Unit), a disk or disk-array and a network interface to the interconnect they are connected.

The actual configuration of a node is transparent (that means not “seen” by the outside user). However there exists a database to describe the system

at the time of the query execution. All relevant data (parameters) describing the architecture are stored in this database. It is assumed that the following architectural characteristics hold during the operation:

- Nodes can perform a dedicated operation (compare, sort and merge two tuples).
- each node has its own mass storage device and the nodes are connected over a network.
- Nodes can send and receive messages.
- The performance of a node will not drop below a given value during query execution.
- The availability of each node during the query execution also is guaranteed.

3 Parallel Sorting Algorithms

In this paper we concentrate on two parallel sorting algorithms, the parallel binary merge sort and the block bitonic sort algorithm and analyze their performance in a generalized multiprocessor and a static simplified grid organization. These two algorithms are broadly accepted and the algorithms of choice in parallel database systems, because of their clear design and well studied properties. Due to the size restriction of the paper only a short description of the two well known algorithms is given in the following. A more detailed and comprehensive discussion can be found in the cited literature.

3.1 Parallel Binary Merge Sort

Binary Merge-Sort uses several phases, to sort mk tuples, where m is the number of pages containing the data set and k denotes the number of tuples per page. We assume that there are much more pages m than processing nodes p (i.e. $m \gg p$) and that the size of the data set is much larger than the available main memory of the nodes.

We assume the very general case that the pages are not distributed equally among the mass storage media of the available nodes and that the tuples are not presorted according to the sorting criteria in the pages. Therefore the algorithm starts with a prepare phase, which distributes the pages equally across all p nodes, sorts the tuples inside every page according to the sort criterion and writes the pages back to disk. After the prepare phase m/p (respectively $m/(p-1)$) pages are assigned to each node and the tuples of each page are sorted. The algorithm continues with the suboptimal phase by merging pairs of longer and longer runs¹. In every step the length of the runs is twice as large as in the preceding run. At the beginning each processor reads two pages, merges them into a run of 2 pages and writes it back to the disk. This is repeated, until all pages are read and merged into 2-pages-runs. If the number of runs exceeds $2p$, the suboptimal phase continues with merging two 2-page-runs to a sorted

¹ A *run* is an ordered (relative to the tuples contained) sequences of pages.

4-page-run. This continues until all 2-page-runs are merged. The phase ends, when the number of runs is $2p$. At the end of the suboptimal phase on each node 2 sorted files of length $m/2p$ exist. In the suboptimal phase the nodes work independently in parallel. Every node accesses its own data only. During the following optimal phase each processor merges 2 runs of length $m/2p$ and pipelines the result (run of length m/p) to a target node. The number of target nodes is $p/2$. The identification of the target-node is calculated by

$$nodenr_{target} = \frac{p}{2} + nodenr_{source}$$

for even source-node-numbers, and

$$nodenr_{target} = \frac{p}{2} + nodenr_{source} + 1.$$

for odd source-node-numbers.

In the postoptimal (last) phase the remaining $p/2$ runs are merged into the final run of length m . At the beginning of the postoptimal phase, we have $p/2$ runs. During this phase one of the p nodes is no longer used. Each of the other nodes is used only once during this phase. Two forms of parallelism are used. First, all nodes of one step work in parallel. Second, the steps of the postoptimal phase overlap in a pipelined fashion. The execution time between two steps consists of merging the first pages, building the first output-page and sending it to the target-node. During the postoptimal phase every node is used only in one step, that means that every node is idle for a certain time.

Thus the algorithm costs are

$$\underbrace{\frac{n}{2p} \log\left(\frac{n}{2p}\right)}_{suboptimal} + \underbrace{\frac{n}{2p}}_{optimal} + \underbrace{+ \log p - 1 + \frac{n}{2}}_{postoptimal} \tag{1}$$

which can be expressed as

$$\frac{n \log n}{2p} + \frac{n}{2} - \left(\frac{n}{2p} - 1\right)(\log p) - 1. \tag{2}$$

3.2 Block Bitonic Sort

Batchers bitonic sort algorithm sorts n numbers with $n/2$ comparator modules in $\frac{1}{2} \log n (\log n + 1)$ steps [10]. Each step consist of a comparison-exchange at every comparator module and a transfer to the target-comparator module. The comparator modules are connected by the perfect shuffle [11]. The perfect shuffle uses three types of comparator modules. The comparator module is represented by a node, which merges two pages and distributes the lower page and the higher page to two target-nodes. The target-nodes are defined by using a mask-information (i.e. the information for the perfect shuffle).

It is necessary to build $2p$ equally distributed and sorted runs of length $m/2p$. The prepare-phase and the suboptimal phase produce the runs. The total cost are:

$$\frac{n}{2p} \left(\log n + \frac{\log^2 2p - \log 2p}{2} \right) \tag{3}$$

4 Parallel Join Algorithms

In this section we introduce two join algorithms for relational databases, a parallel "nested-loop" and a parallel "sort-merge" algorithm. Like the sort algorithms in section 3, the presented join algorithms are commonly used in database systems.

4.1 Nested-Loop Join

The inner (smaller) relation T , and the outer relation R (larger one) are joined together. The algorithm can be divided in two steps:

1. **Initiate**

Each of the processors read a different page of the outer relation R

2. **Broadcast and join**

All pages of the inner relation T are sequentially broadcasted to the processors. After receiving the broadcasted page, each processor joins the page with its page from R .

n and m are the sizes (number of pages) of the relations R and R' , and we suppose $n \geq m$. To perform the join of R and R' we assign p processors. If $p = n$, the execution time is:

$$\begin{aligned} T_{nested-loop} = & T(\text{read a page of } R) \\ & +mT(\text{broadcast a page of } R') \\ & +mT(\text{join 2 pages}) \end{aligned} \tag{4}$$

S is the join selectivity factor and indicates the average number of pages produced by the join of a single page of R with a single page of R' . Joining two pages is performed by merging the pages, sorting the output page on the join attribute and write the sorted page to disk.

$$S = \frac{size(R \text{ join } T)}{mn} \tag{5}$$

If the number of processors p smaller than the number of pages n , step 1) and 2) must be repeated $\frac{n}{p}$ times. Therefore the costs for the Parallel Nested-Loop Join is

$$T_{nested-loop} = \frac{n}{p} \left[C_r + m[C_r + C_m + S(C_{so} + C_w)] \right] \tag{6}$$

4.2 Sort Merge Join

The algorithm processes in two steps. The first step of the algorithm is to sort the two relations on the join attribute (we assume, that the two relations are not already sorted). After sorting, the second step is performed, where the both sorted relations are joined together and the result relation is being produced. If we use the block bitonic sort in the first step described in section 3, the costs of the Sort Merge Join are

$$T = \left[\frac{n}{2^p} \log n + \frac{m}{2^p} \log m + (\log^2 2p - \log 2p) \frac{n+m}{4p} \right] C_p^2 + (n+m)C_r + \max(nm)C_m + mnS(C_{so} + C_w). \quad (7)$$

Using the Parallel Binary Merge Sort the costs in term of C_2^p costs are

$$T = \left[\frac{n \log n}{2^p} + \frac{n}{2} - \left(\frac{n}{2^p} - 1 \right) (\log p) - 1 \right] C_p^2 + (n+m)C_r + \max(nm)C_m + mnS(C_{so} + C_w). \quad (8)$$

4.3 Analysis Parameters

For comparing the algorithm we use the same definitions of the analysis parameter as described in [1].

We define with n the number of pages with k tuples each, and with p the number of processors.

Communication Cost. A processor must request a page, for this purpose a message is necessary, the cost for such an "I/O-related" message is C_{msg}

I/O Cost Parameters

- H ... certain hit ratio for the cache
- H' ... fraction amount of time a free page frame will be available in the cache during a write operation
- R_c ... cost of a cache to processor transfer
- R_m ... cost of a mass-storage transfer
- k is the number of tuples in a page,
- C are costs of a simple operation (scan, compare, add)

The average cost of a read by a processor is

$$C_r = HR_c + (1 - H)(R_c + R_m) + 2C_{msg}$$

The average cost of writing a page is

$$C_w = H'R_c + (1 - H')(R_c + R_m) + 2C_{msg}$$

Merge Cost

- V are costs of moving a tuple inside a page

Thus the costs of merging a page are

$$C_m = 2k(C + V)$$

To group some of the above parameters we define analogously to [1] a "2-page operation" C_p^2 as

$$C_p^2 = 2C_r + C_m + 2C_w$$

5 Analysis

The costs of reading a page C_r in a simplified Grid architecture can be split into two parts:

- C_{ard} average costs of reading a page from the disk to the main memory of the remote node, and
- C_{arn} average costs of transferring a page from the remote node to the local node.

Analogously to reading a page, writing a page C_w can also be split in two parts

- C_{awn} average costs of transferring a page from the local node to the remote node, and
- C_{awd} average costs of writing a page from the main memory to the disk of the remote node.

5.1 Lessons Learned for Grid Workflow Orchestration

Based on a thorough analysis, which is beyond the scope of this paper, we proved the correlation "the higher the network speed, the lower the impact on the costs of a C_p^2 operation", which is also intuitively clear. The costs of disk accesses are therefore much less influencing the overall performance than the network costs. Focusing on the sort algorithms the impact on the performance of the sort merge algorithms depends predominantly on the network bandwidth. Therefore the nodes with the best network-bandwidth numbers should be grouped to perform the last (*postoptimal_{II}*) phase in the binary merge sort. One stage in this last phase consists of a number of sender and a (number of) receiver nodes. The algorithm of defining the layout of the workflow for the postoptimal phase can be described by choosing the nodes with the best network bandwidth starting from the final stage. If the bandwidth is the same for some nodes the ones with the best computational power have to be chosen then. This can be described by the following algorithm:

1. Determine the network bandwidth and processing power for each processing node
2. Sort nodes according to network bandwidth

3. Nodes with equal network bandwidth in the sequence are sorted according to their processing power
4. Identify postoptimal phase as binary tree structure with node creating final run of length m as root
5. Starting from the beginning of sequence (i.e. best node first) map nodes level-wise from right to left beginning from root (root is level 0, successors of root are level 1, etc.).

5.2 Effects on the Performance of the Sort Algorithms

Based on the work of Bitton et al. in a generalized multiprocessor organization the block bitonic sort always outperforms the binary merge sort (see Figure 3 which is based on the analysis in [1]). To analyze the mapping of the algorithms onto a simplified Grid organization we have specifically pay attention to the three phases of the algorithms as laid out in section 3. The last phase (postoptimal) of the Binary Merge Sort algorithm is split into three parts (see equation 9) because only one processor is necessary for $\frac{n}{2}$ costs.

$$\underbrace{\left[\log p - 1 + \frac{n}{2} \right]}_{\text{postoptimal}} \quad \rightarrow \quad \underbrace{\log p - 1}_{\text{postoptimal}_I} \quad + \quad \underbrace{\frac{n}{2}}_{\text{postoptimal}_{II}} \quad (9)$$

If we choose for this "bottleneck" the nodes of the Grid with the best network bandwidth available, the effect on the overall performance has to be at least remarkable. We specifically emphasize that even only one processing node with high network performance is worth while to exploit this effect. It is intuitively clear that this situation can be seen as normal in a heterogenous Grid organization, where nodes with different performance characteristics are the rule.

This leads to the clear policy for orchestration of a Grid workflow for a parallel binary merge sort to use nodes with the highest network performance in the postoptimal_{II} phase as laid out in algorithm of section 5.1.

On the other hand using one high performance node in the bitonic sort gives no performance at all, because this node is slowed done by all other nodes working in parallel in a staged manner.

This effect that the binary merge sort now outperforms the block bitonic sort in a simplified Grid organization is shown in the Figure 4 by the line labeled "binary merge modified" (please notice the logarithmic scale of the values).

This effect can easily be explained by Amdahl's law too, where simply said the performance of a parallel algorithm is dependent on its sequential part. More specific Amdahls's law is stating that the speedup is limited by the sequential part. This is intuitively clear that even the most powerful node will limit the performance increase if the number of used nodes for the whole parallel algorithm is increasing. A speedup and scale-up analysis will clear up this issue.

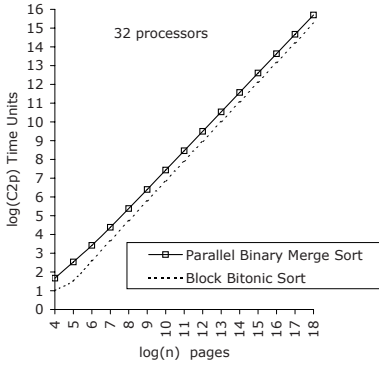


Fig. 3. Sort in a Generalized Multiprocessor Organization

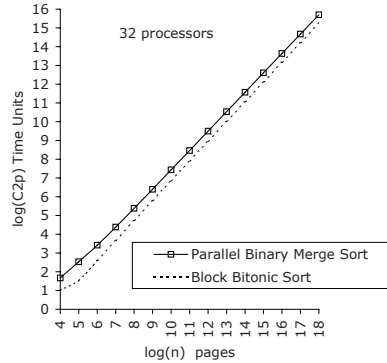


Fig. 4. Sort in a Simplified Grid Organization

5.3 Effects on the Performance of the Join Algorithms in a Generalized Multiprocessor Organization

Similar to the effects on the performance of the Sort Algorithms in a Generalized Multiprocessor Organization, the Merge-Sort Join algorithm with the block bitonic sort outperforms the Nested-Loop Join and also outperforms the Merge-Sort Join based on the binary-merge sort algorithm, unless the number of processors available is close to the larger relation size. Figure 5 shows the join algorithms with a selectivity factor of 0,001. If the ratio between the relation sizes is significantly different from 1, the nested-loop algorithm outperforms the merge-sort (except for a small numbers of processors). For lower selectivity values, the merge-sort algorithm performs better than the nested-loop algorithm because the merge step (handled by a single processor) has to output fewer pages.

In a generalized multiprocessor organization we have analyzed the algorithms with a selectivity factor between 0,001 and 0,9. The effect is, that the difference between the two merge-sort algorithms stays constant (if $S \geq 0.05$). The reason is, that the merge costs are only marginal to the overall costs of the algorithm.

5.4 Effects on the Performance of the Join Algorithms in a Simplified Grid Organization

In the simplified grid organization the Merge-Sort join algorithm based on the bitonic-sort outperforms the Merge-Sort join based on the binary merge sort up to 2^6 processors, see "Parallel Merge-Sort Join (Binary Merge Sort) modified" in Figure 6.

As in the multiprocessor organization we have analyzed the join algorithms in the simplified grid organization with a selectivity factor between 0,05 and 0,9. The effect is the same for the generalized multiprocessor organization, the difference between the merge-sort algorithm (Parallel Merge-Sort Join (Bitonic)

and Parallel Merge-Sort (Binary Merge)) remains constant (if $S \geq 0.05$). The reason is, that the merge costs have insignificant influence on the overall costs of the algorithm. The Parallel Merge-Sort (Binary Merge) modified is always faster (in terms of C_2^p costs) than the unmodified version.

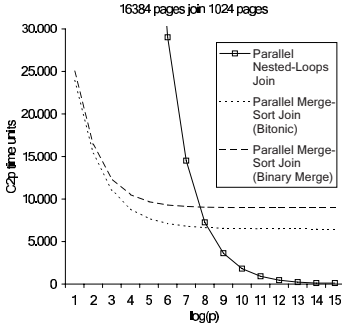


Fig. 5. Join in a Generalized Multi-Proc. Organization with $S=0,001$

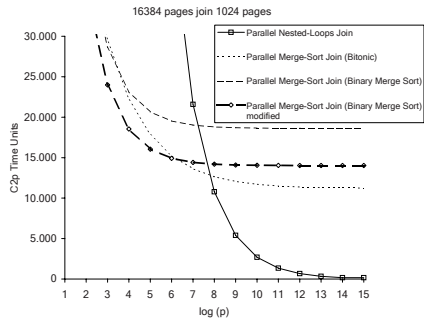


Fig. 6. Join in a Simplified Grid Organization with $S=0,001$

6 Conclusion and Future Work

In this paper the performance of the well-known parallel Sort-merge, Bitonic sort, Nested-Loop join and Merge-sort join algorithms for parallel database systems was analyzed. We developed an analytical evaluation model based on a simplified Grid architecture.

These are important results for the creation of parallel execution plans for database queries in the Grid. This work is performed as part of a project, which aims towards the development of a query optimizer for database query execution workflows and their orchestration in the Grid. We aim for a general execution time model for parallel query evaluations which can be integrated into a smart broker. This should give us the basis to create (near) optimal bushy query execution plans, which exploit operator- and data-parallelism in a Grid environment. In a further state of this research project we also will take into a account the availability of nodes and will extend our static model towards a more dynamic version.

References

1. Bitton, D., Boral, H., DeWitt, D.J., Wilkinson, W.K.: Parallel algorithms for the execution of relational database operations. *ACM Trans. Database Syst.* 8, 324–353 (1983)
2. Online: The gridbus project: grid computing info centre (grid infoware). Website (last access: 2007-01-30)

3. Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., Stockinger, K.: Data management in an international data grid project. In: IEEE/ACM International Workshop on Grid Computing Grid, ACM Press, New York (2000)
4. Antonioletti, M., Atkinson, M., Baxter, R., Borley, A., Hong, N.C., Collins, B., Hardman, N., Hume, A., Knox, A., Jackson, M., Krause, A., Laws, S., Magowan, J., Paton, N.W., Pearson, D., Sugden, T., Watson, P., Westhead, M.: The design and implementation of grid database services in ogsa-dai. *Concurrency and Computation: Practice and Experience* 17, 357–376 (2005)
5. Gounaris, A., Sakellariou, R., Paton, N.W., Fernandes, A.A.A.: Resource scheduling for parallel query processing on computational grids. In: 5th International Workshop on Grid Computing (GRID 2004), pp. 396–401 (2004)
6. Soe, K.M., Aung, T.N., Nwe, A.A., Naing, T.T., Thein, N.: A framework for parallel query processing on grid-based architecture. In: ICEIS 2005, Proceedings of the Seventh International Conference on Enterprise Information Systems, pp. 203–208 (2005)
7. Iyer, B., Dias, D.: Issues in parallel sorting for database systems. In: Proc. Int. Conf. on Data Engineering, pp. 246–255 (1990)
8. Jarke, M., Koch, J.: Query optimization in database systems. *ACM Computing Surveys* 16, 111–152 (1984)
9. Bitton, D., DeWitt, D., Hsiao, D., Menon, J.: A taxonomy of parallel sorting. *ACM Computing Surveys* 16, 287–318 (1984)
10. Batcher, K.E.: Sorting networks and their applications. In: Proc. of the 1968 Spring Joint Computer Conference, Atlantic City, NJ, April 30-May 2, vol. 32 (1968)
11. Stone, H.: Parallel processing with the perfect shuffle. *IEEE Trans. Computing* C-20 (1971)