Parallel Database Processing on a 100 Node PC Cluster: Cases for Decision Support Query Processing and Data Mining

Takayuki Tamura, Masato Oguchi, Masaru Kitsuregawa

Institute of Industrial Science, The University of Tokyo
7-22-1 Roppongi, Minato-ku, Tokyo 106, Japan
Phone: +81-3-3402-6231, Fax: +81-3-3423-2834
{tamura,oguchi,kitsure@tkl.iis.u-tokyo.ac.jp}

Abstract

We developed a PC cluster system consists of 100 PCs. Each PC employs the 200MHz Pentium Pro CPU and is connected with others through an ATM switch. We picked up two kinds of data intensive applications. One is decision support query processing. And the other is data mining, specifically, association rule mining.

As a high speed network, ATM technology has recently come to be a de facto standard. While other high performance network standards are also available, ATM networks are widely used from local area to widely distributed environments. One of the problems of the ATM networks is its high latencies, in contrast to their higher bandwidths. This is usually considered a serious flaw of ATM in composing high performance massively parallel processors. However, applications such as large scale database analyses are insensitive to the communication latency, requiring only the bandwidth.

On the other hand, the performance of personal computers is increasing rapidly these days while the prices of PCs continue to fall at a much faster rate than workstations'. The 200MHz Pentium Pro CPU is competitive in integer performance to the processor chips found in workstations. Although it is still weak at floating point operations, they are not frequently used in database applications.

Thus, by combining PCs and ATM switches we can construct a large scale parallel platform very easily and very inexpensively. In this paper, we examine how such a system can help the data warehouse processing, which currently runs on expensive high-end mainframes and/or workstation servers.

In our first experiment, we used the most complex query of the standard benchmark, TPC-D, on a 100 GB database to evaluate the system compared with commercial parallel systems. Our PC cluster exhibited much higher performance compared with those in current TPC benchmark reports. Second, we parallelized association rule mining and ran large scale data mining on the PC cluster. Sufficiently high linearity was obtained. Thus we believe that such commodity based PC clusters will play a very important role in large scale database processing.

1 Introduction

Recently, massively parallel computer systems are moving away from proprietary hardware components to commodity parts for CPUs, disks, etc. However, they still employ proprietary interconnection networks for higher bandwidths and lower latencies. Though the bandwidths of the upcoming standards for high speed communication such as ATM are being improved rapidly due to the recent progress in communication technologies, the latencies will remain high. This can be considered a serious flaw of the ATM networks in composing high performance massively parallel processors. However, there are applications which are insensitive to the communication latency: large scale database analyses. In these applications, a large amount of data are exchanged among the processing nodes. But the communication is one-way, thus requiring only the bandwidth.

On the other hand, the performance of personal computers is increasing rapidly these days. For example, the 200MHz Pentium Pro CPU is competitive in integer performance to the processor chips found in workstations, although it is still weak at floating point operations, which are not frequently used in database applications. In addition, the prices of PCs continue to fall at a much faster rate than workstations'. In 1996, more than ten million PCs were sold resulting in a much better performance/price ratio for PCs than for workstations.

Thus looking over recent technology trends, ATM connected PC clusters are very promising approach for massively parallel database processing, which we believe to be very important in addition to the conventional scientific applications. Considerable research on parallel database processing has been done in the last decade[7]. We have developed several parallel hardware platforms in addition to the parallel operating systems and the parallel relational algorithms[10, 11, 16]. Through the research, the effectiveness of parallelization has been proved, and several vendors are now shipping parallel DBMS products running on commercial parallel platforms. However, all of such platforms are either SMP machines such as Sun's UltraEnterprise or proprietary hardware such as IBM's SP-2 and NCR's WorldMark. The performance of parallel database processing on commodity based large scale PC clusters has never been reported. Recently, the Berkeley NOW team reported the performance of parallel sorting operations on their workstation cluster[3]. Though their approach is quite attractive, the application is quite simple and does not reflect the many aspects in the real relational database query processing. Moreover, their system is not so inexpensive as commodity based PC clusters.

As for PC cluster development, various research projects have been performed, in which some scientific benchmarks are executed on clusters [15, 6]. However, absolute performance of such clusters is not attractive compared with massively parallel processors because of insufficient performance of PCs and networks used in the projects, though preferably good price/performance has been achieved.

We developed a PC cluster system consisting of 100 PCs for parallel database processing. The PCs employing the Pentium Pro CPU are connected through an ATM switch. In this paper, we picked up two kinds of data intensive applications. The first is decision support query processing. The most complex query of the standard benchmark, TPC-D, on a 100 GB database was used to evaluate the system compared with commercial parallel systems. Our PC cluster exhibited much higher performance compared with those systems found in current TPC benchmark reports. The second is association rule mining. We developed a parallel algorithm for this application. As a result, sufficiently high linearity was obtained.

In section 2, our PC cluster system is presented. In section 3, experiments on decision support query processing are described. Section 4 explains parallel association rule mining. Our concluding discussions are given in section 5.

Figure 1: PC cluster pilot system

CPU	Intel Pentium Pro 200MHz
Chip Set	Intel 440FX
Main Memory	64MB
Disk Drives	For OS Western Digital Caviar 32500 (EIDE, 2.5GB)
	For databases Seagate Barracuda (Ultra SCSI, 4.3GB, 7200 RPM)
SCSI HBA	Adaptec AHA 2940 Ultra Wide
ATM NIC	Interphase 5515 PCI ATM Adapter
OS	Solaris2.5.1 for x86

Table 1: Configuration of each PC

2 100 node PC cluster system

2.1 System components

Our PC cluster pilot system consists of one hundred 200MHz Pentium Pro PCs, connected by a 155 Mbps ATM network as well as by a 10 Mbps Ethernet network. Figure 1 shows a picture of the system. Table 1 shows the configuration of each PC node. We use the RFC-1483 PVC driver for IP over ATM[9][12]. Hitachi AN1000-20, which has 128 UTP-5 ports, is used as an ATM switch.

2.2 Fundamental performance characteristics

First, we measured fundamental performance of network communication and disk accesses. Figure 2 shows the effective throughput of point-to-point communication, using the socket interface for TCP/IP over ATM. 133MHz Pentium PCs were also tested for the sake of comparison. Contrary to our expectation, the throughput between 200MHz Pentium Pro PCs is sufficiently high in spite of using ‘‘heavy’’ TCP/IP. It exceeds 110Mbps when we choose the appropriate message size such as 8KBytes. As for 133MHz Pentium PC, the CPU really suffers from the protocol overhead, resulting in much lower performance. Based on the result, we decided to employ the standard TCP/IP as the communication protocol within the cluster, instead of tackling development of a new light-weight protocol. On the other hand, round trip

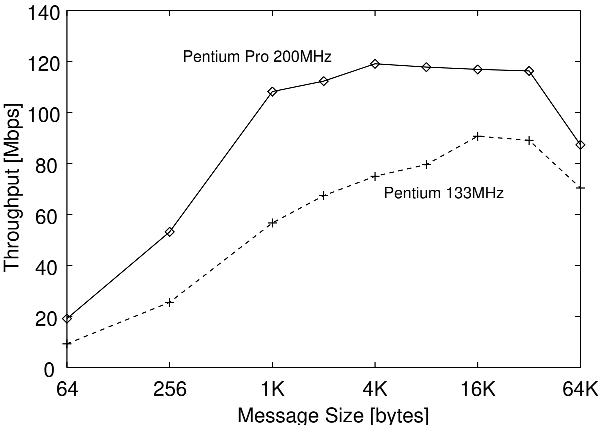

Figure 2: Throughput of point-to-point communication

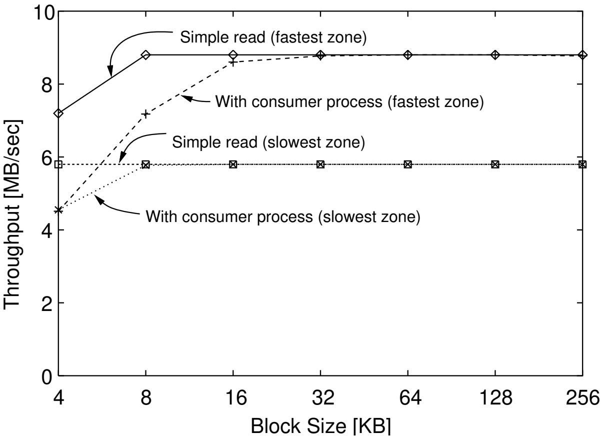

Figure 3: Throughput of sequential disk read

average latency of 1 byte messages is 448 μsec, which is quite large compared with that of massively parallel processors. However, this does not affect the performance of database analyses so much, because such applications require mainly one-way communication.

Next, we measured the sequential read performance of the SCSI disk. The disk is accessed via the raw device interface, which bypasses the buffer cache mechanism of the Unix file system. For modeling different CPU loads, two kinds of tests were performed. In the first test, `read` system calls are issued consecutively without processing the data. ‘‘Simple read’’ curves in figure 3 show the results of this test. Due to the zone bit recording technique, throughput of this disk varies considerably depending on the location of the blocks to be accessed. ‘‘Fastest zone’’ and ‘‘slowest zone’’ in the graph show the upper bound and the lower bound, respectively, of the results. In the second test, read data are check-summed before being discarded, to model the CPU load in actual database processing. To access the disk asynchronously, two threads, one for disk read and another for check summing, are created using the POSIX thread library supported by Solaris operating system. They communicate through a queue of buffers. ‘‘With consumer process’’ curves in figure 3 show the results of this test. When the CPU is heavily loaded, blocks smaller than 32KBytes are no more adequate for disk accesses due to the system call overhead. Thus we chose 64KBytes for the disk I/O block size. In this case, the throughput is between 5.8 MB/sec and 8.8 MB/sec.

3 Decision support query processing

3.1 Parallel hash join operations

The widespread adoption of relational database systems and advances in on-line transaction processing techniques have brought very large databases and transaction logs. Users are now interested in analyzing statistics of the databases to make their new business plans. The applications for such analyses include decision support, market analysis, sales trend analysis, and data mining, and database servers for supporting these applications are called ‘‘data warehouses.’’ The standardization of the database benchmarks reflects the prevalence of such applications. TPC Benchmark^{TM}A, B, and C are well known benchmarks for transaction processing, and frequently used as performance metrics for database systems [8]. In 1995, TPC announced yet another benchmark named TPC Benchmark^{TM}D[17], which is targeted for decision support queries. These queries are ad-hoc and intensively use heavy relational operators such as join, sort, and aggregation. Thus very high performance relational database systems which can efficiently scan a large amount of data and have sophisticated processing algorithms for heavy operators are required.

Join is one of the most expensive relational operators, to combine multiple relations according to join conditions. The hash based join algorithms are well suited for parallel processing and have been widely researched. The parallel hash join algorithm first partitions input relations into disjoint subsets called buckets, by applying a hash function to the join attribute of each tuple. Since each bucket consists of the tuple which map to the same hash value, joining the two relations results in joining each pair of the corresponding buckets from the two relations. This means each join operation can be performed in parallel among the processing nodes. The outline of the parallel hash join algorithm is as follows:

1. Build Phase:

   Each processing node applies the hash function to each tuple of its portion of one of the two input relations, R. Then the tuples are sent out to their destinations according to their hash values. Each received tuple is inserted into a hash table on each processing node.

2. Probe Phase:

   The other relation S is read out from the disks and each tuple is sent to the destination node according to its hash value. The hash table on the processing node is probed with the received tuples to produce result tuples.

More than two relations can be joined together within a query. A join of N relations requires $N - 1$ joins. The order in which the joins are performed is determined by the query optimizer, and a variety of query execution plans have been proposed[13]. One of such plans is the right-deep multi-join, which begins by building $N - 1$ hash tables in memory from $N - 1$ relations before probing them successively in a pipeline manner with tuples of the remaining relation.

3.2 Software architecture for query processing

So far we have developed several parallel hardware platforms in addition to the parallel operating systems and the parallel relational algorithms[10, 11, 16]. The SDC, the super database computer, is one of such platforms and is a cluster of shared memory multiprocessor nodes, and the purpose of the project was to examine the viability of microprocessor based parallel query processing servers. Throughout these projects, we developed proprietary architectures based on commodity microprocessors. Because the commercial workstations in those days showed poor I/O performance, we didn’t employ them as

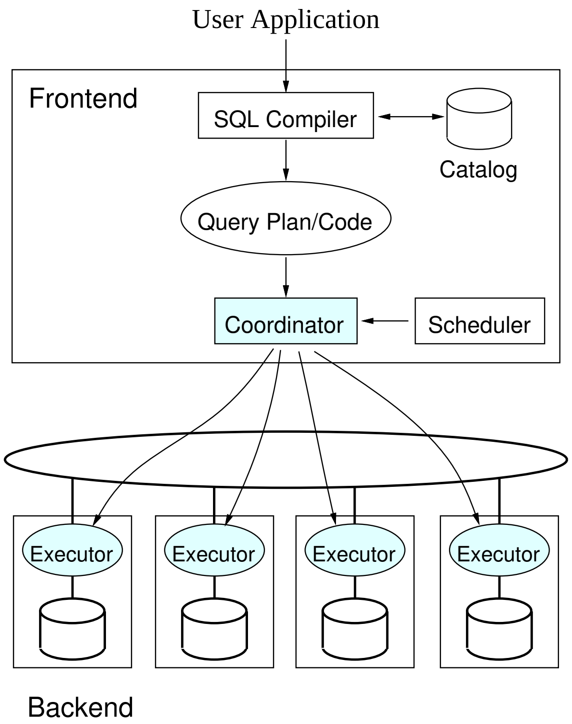

Figure 4: Global architecture of database server

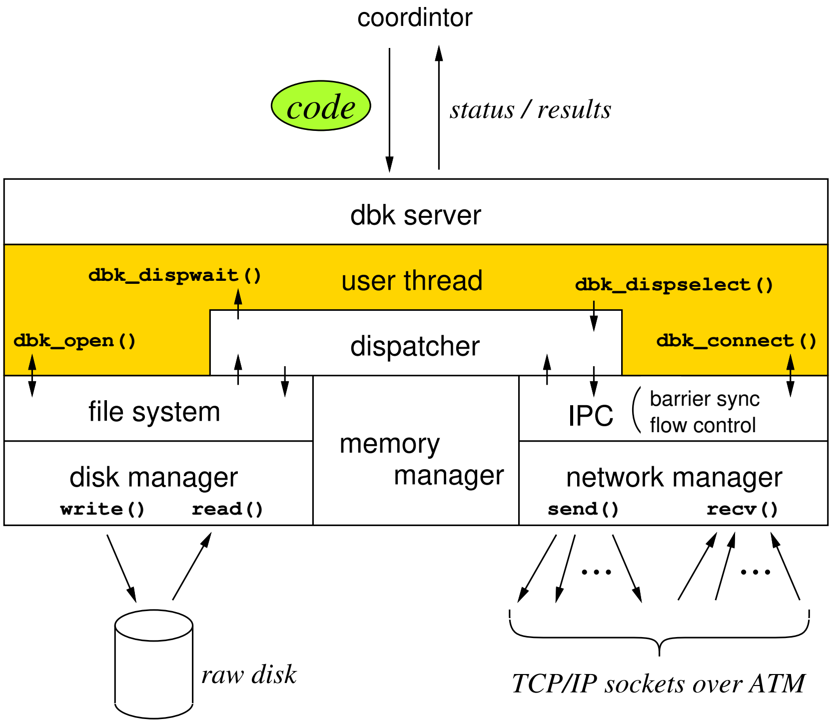

Figure 5: Structure of database kernel at each node

processing elements. Standard LAN technologies were also immature so that we were forced to design and implement our own interconnection networks.

Now that the circumstances have dramatically changed, it is reasonable to use commodity PCs as processing elements and ATM technology for interconnection. With ATM connected PC clusters, we have only to develop software for parallel query processing. So we ported the system software developed for the SDC to our PC cluster system. Currently, we emulate the device driver layer of the SDC code by user level threads (POSIX thread) on Solaris 2.5.1 operating system. We are also planning kernel level implementation for maximal efficiency.

Figure 4 depicts the global architecture of our database server. Queries issued from the user applications are compiled and optimized at the front-end machine to produce the executable code. We take the compiled mode, where the generated code consists of the processor's native code, instead of the interpreted mode for efficiency and flexibility. Currently, however, the SQL compiler and optimizer are not yet ready, and the work has to be done by hand. That is, we determine a plan for the query and write a C code for the plan.

Execution of a query is initiated by the coordinator program. The coordinator broadcasts the generated code to each PC node, where a server process for query execution is running. Figure 5 shows the structure of the server process, called database kernel. The database kernel consists of several permanent threads, and one of such threads, named the dbk server, interacts with the coordinator. It creates a user thread for executing the code passed from the coordinator, and monitors exceptional events from the user thread or the coordinator. The exceptional conditions from the user thread are reported to the coordinator and cause user threads running at other nodes to abort. The generated code contains only parts specific to the corresponding query, such as the initialization and finalization code according to the execution plan, and evaluation code for predicates in the query. Commonly used code such as hash table manipulation and various conversion routines are contained in the database kernel, and are dynamically linked to the target code before its execution. Each of the user threads then sets up the I/O streams and registers an operator to each input stream as a callback function. Another permanent thread, dispatcher, keeps polling all the registered input streams and evaluates callback functions with data delivered by I/O threads for disk and network. Thus, most of the processing is done within the context of the dispatcher, and the user thread suspends until the EOF condition is met. This model of ‘‘centralized I/O’’ is designed so as to support multiple queries running at the same time.

Once execution starts, the PCs perform their operations without any centralized control. Barrier synchronizations are encapsulated within open and close operations of network connections to simplify the code. On completion of the network open operation (`dbk_connect()`), all the nodes are guaranteed to be ready for receiving data, and on completion of the network close operation (`dbk_shutdown()`), it is guaranteed that no more data will arrive.

3.3 Performance evaluation using a TPC-D benchmark query

TPC-D benchmark consists of 17 complex queries issued to a database which contains eight relations[17]. Among them, the most time consuming query is the query 9, which requires five join operations. Figure 6 shows SQL description of the query. Here, we examine the execution times of this query with different plans. We generated a 100GB test database, shown in table 2, using a modified version of the `dbgen` program, and partitioned tuples of each relation horizontally over the 100 disks according to the hash values of the primary key. In fact, NATION and REGION are not partitioned but replicated because they are quite small. We didn't create any indexes to force the full scan of the relations, because the indexes wouldn't help ad-hoc queries.

Figure 7 depicts an execution plan of the query 9 with the right-deep tree. First, four hash tables are

```
select Nation, Year, sum(Amount) as Sum_Profit
from (select N_Name as Nation,
        extract(year from O_Orderdate) as Year
        L_Extendedprice * (1 - L_Discount)
          - PS_Supplycost * L_Quantity
            as Amount
      from part, supplier, lineitem,
           partsupp, orderx, nation

      where  S_Suppkey  = L_Suppkey
         and PS_Suppkey = L_Suppkey
         and PS_Partkey = L_Partkey
         and P_Partkey  = L_Partkey
         and O_Orderkey = L_Orderkey
         and S_Nationkey = N_Nationkey
         and P_Name like ’%green%’
     )
group by Nation, Year
order by Nation, Year desc
```

Figure 6: SQL description of TPC-D query 9

built from relations PART, PARTSUPP, SUPPLIER, and ORDER, and then they are probed in sequence by LINEITEM. Because only a part of the attributes are necessary for the query, the tuples shrink so much that whole hash tables can fit in memory, while all tuples are selected from three of the relations. Σ in the plan represents the aggregation (the sum by group), which is first performed independently at each node before gathering the results at the master node (node #0) to lower the network traffic. The final join of the aggregation results and NATION is performed locally at the master node, because NATION is too small for parallelization.

Figure 8 shows a trace of the CPU utilization and the effective throughput of the disk and the network at the master node during execution of this plan. In the build phases from the phase #1 through the phase #4, the CPU utilization was less than 30%, and the disk throughput reached near the maximum value,

Relation	Number of Tuples	Tuple length
SUPPLIER	1,000,000	208 bytes
PART	20,000,000	176 bytes
PARTSUPP	80,000,000	220 bytes
CUSTOMER	15,000,000	236 bytes
ORDER	150,000,000	140 bytes
LINEITEM	600,037,902	156 bytes
NATION	25	192 bytes
REGION	5	188 bytes

Table 2: Database for 100GB TPC-D benchmark

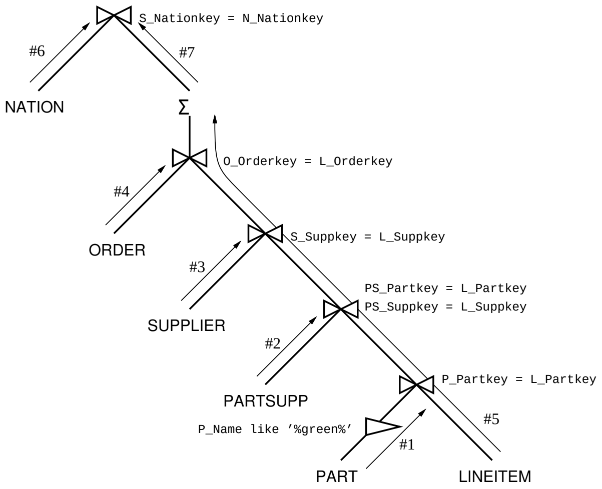

Figure 7: Execution plan of TPC-D query 9 with right-deep tree

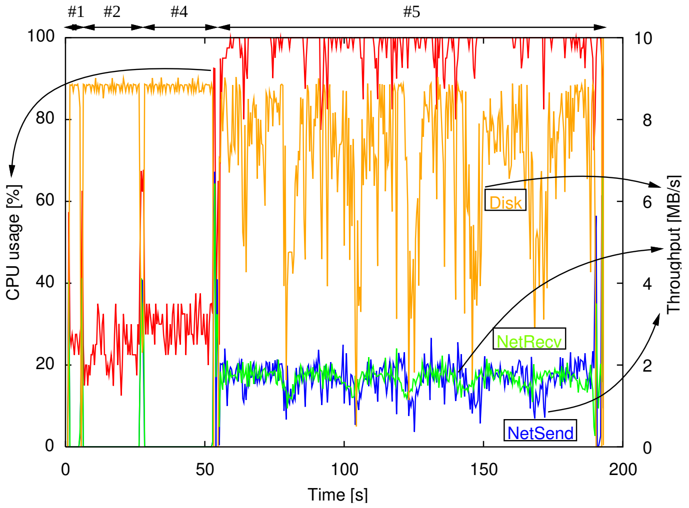

Figure 8: Execution trace of TPC-D query 9 with right-deep tree

8.8 MB/sec, resulting in the disk I/O bound. However, in the probe phase #5, the CPU load increased to 100%, with the disk throughput decreasing. The reason of this transition to the CPU bound is due to the heavy CPU load required for processing multiple probe operations concurrently.

Figure 9 depicts another execution plan with the left-deep tree, which does not involve concurrent probe operations. In this case, the results of each probe operation turn into the hash table for the next probe operation. Figure 10 shows an execution trace of this plan. Except for the phase #4, the disk throughput reached near the maximum. In the phase #5, the CPU load increased to near 70% because the aggregation was performed concurrently with the probe operation by ORDER, but the bottleneck was still in the disk I/O. In the phase #4, though the CPU was heavily loaded, higher disk throughput was obtained compared with the phase #5 of the right-deep plan. The difference of the disk throughput between these phases resulted in the different elapsed times because the same relation LINEITEM was accessed in these phases. The elapsed time of the phase #5 of the right-deep plan was 140 seconds, while that of the phase #4 of the left-deep plan was 123 seconds.

In the above experiments, execution times are dominated by the disk read times of input relations.

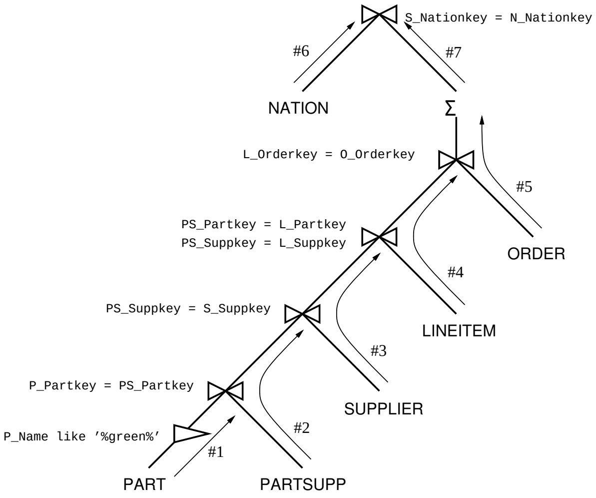

Figure 9: Execution plan of TPC-D query 9 with left-deep tree

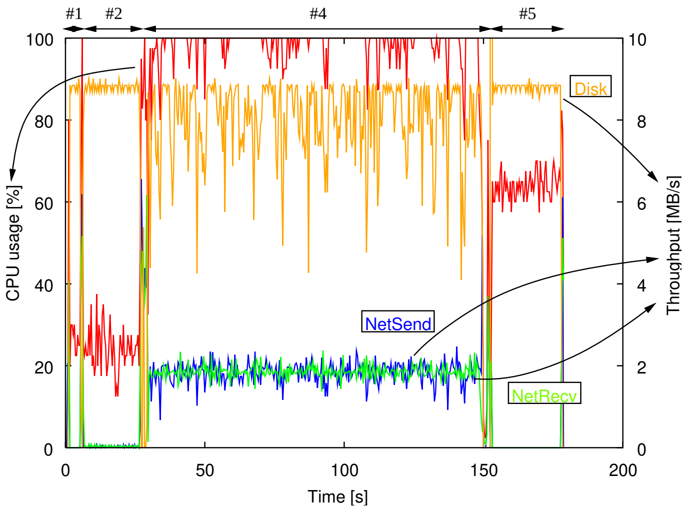

Figure 10: Execution trace of TPC-D query 9 with left-deep tree

Though the execution plans affect the performance of query processing, the effect is rather small unless extra I/Os are incurred. Because the disks are providing data at a high rate near the maximum of 8.8 MB/sec, we have to reduce the amount of I/O in order to obtain further speedup. Without indexes, which are not so helpful to this query, this can be accomplished by avoiding extra data transfer of unnecessary attributes in each relation. For this purpose, we store each attribute of a relation in an individual file, separately from others. This physical storage organization is called transposition, or vertical partitioning[4, 5].

Figure 9 shows an execution plan of the query 9 for the transposed file organization. Due to the increase in number of input files, the execution tree gets rather complicated. The join operations in the tree can be classified into two types: inter-relation joins which are essential to the query, and intra-relation joins (tuple ID joins) for reconstructing original tuples (in projected forms) from disconnected attributes. The former appear as red links in the figure, and the latter as light blue. However, we omit the further details here. Figure 12 shows an execution trace of this plan. Throughout the execution, the CPU utilization stayed almost 100while the disk throughput dropped significantly, meaning that the bottleneck turned from I/O to CPU. Because of this heavy CPU load, the execution time wasn't reduced proportionally to the amount

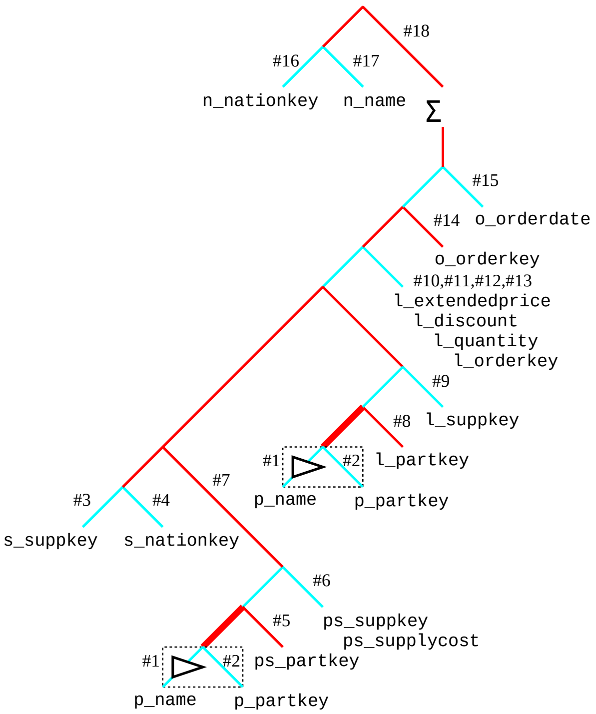

Figure 11: Execution plan of TPC-D query 9 for transposed files

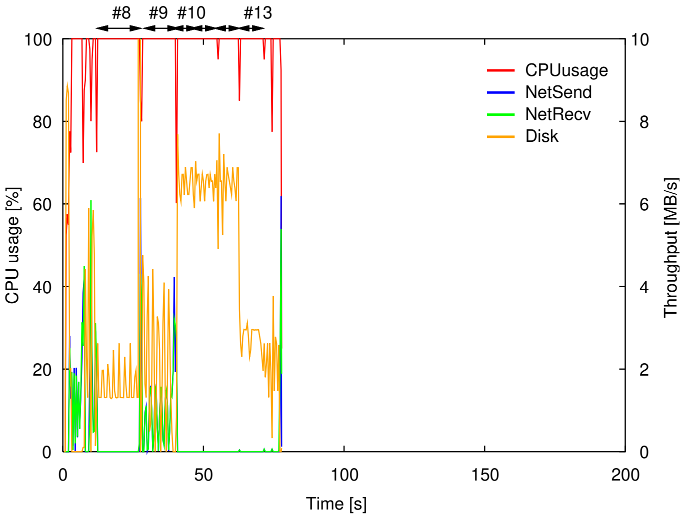

Figure 12: Execution trace of TPC-D query 9 with transposed files

of I/O. But the resulting speedup compared with the previous plans exceeds 2, which is quite satisfactory.

Table 3 shows the results of above right-deep (rd), left-deep (ld), and transposed file (tp) methods, along with the reported results of other commercial systems for 100GB TPC-D query 9. Because our system lacks the software and maintenance price metrics, the overall system price can't be determined accurately. Hardware components themselves cost less than $0.5M. We can observe that our system achieves fairly good performance. Above all, the execution time with the transposed files is twelve times as short as the most powerful commercial platform. These results strongly support the effectiveness of the commodity PC based massively parallel relational database servers.

System	Exec. Time [s]	Price
Teradata on NCR 5100M 160 × 133MHz Pentium 20GB Main Memory 400 Disk Drives	953.3	$17M
Oracle 7 on DEC AlphaServer 8400 12 × 437MHz DECchip 21164 24GB Main Memory 84 Disk Drives	1884.9	$1.3M
Oracle 7 on SUN UE6000 24 × 167MHz UltraSPARC 5.3GB Main Memory 300 Disk Drives	2639.3	$2.1M
IBM DB2 PE on RS/6000 SP 306 96 × 112MHz PowerPC 604 24GB Main Memory 96 Disk Drives	2899.4	$3.7M
Oracle 7 on HP9000 EPS30 12 × 120MHz PA7150 3.75GB Main Memory 320 Disk Drives	7154.8	$2.2M
Our Pilot System 100 × 200MHz Pentium Pros 6.4GB Main Memory 100 Disk Drives	(rd) 193.7 (ld) 177.2 (tp) 77.1	(see text)

Table 3: Execution time of 100 GB TPC-D Q9 on several systems

4 Data mining

4.1 Association rule mining

Data mining, which is a recent hot research topic in the database field, is a method of discovering useful information, such as rules and previously unknown patterns, existing behind data items. It enables more effective utilization of transaction log data, which have been just archived and abandoned. Among the major applications of data mining is association rule mining, so called ''basket analysis.'' Each of the transaction data typically consists of a set of items bought in a transaction. By analyzing them, one can derive some association rule such as, ''90% of the customers who buy both A and B also buy C.'' In order to improve the quality of obtained rules, a very large amount of transaction data have to be examined, requiring quite a long time to complete.

First we introduce some basic concepts of association rule. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items, and $\mathcal{D} = \{t_1, t_2, \ldots, t_n\}$ be a set of transactions, where each transaction t_i is a set of items such that $t_i \subset \mathcal{I}$. An itemset X has *support* s in the transaction set $\mathcal{D}$ if s% of transactions in $\mathcal{D}$ contain X, here we denote $s = support(X)$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X, Y \subset \mathcal{I}$, and $X \cap Y = \emptyset$. Each rule has two measures of value, *support* and *confidence*. The *support* of the rule $X \Rightarrow Y$ is $support(X \cup Y)$. The *confidence* c of the rule $X \Rightarrow Y$ in the transaction set $\mathcal{D}$ means c% of transactions in $\mathcal{D}$ that contain X also contain Y, which can be written as $support(X \cup Y)/support(X)$.

For example, let $T_1 = \{1, 3, 4\}$, $T_2 = \{1, 2, 3, 5\}$, $T_3 = \{2, 4\}$, $T_4 = \{1, 2\}$, $T_5 = \{1, 3, 5\}$ be the transaction database. Let *minimum_support* and *minimum_confidence* be 60% and 70% respectively. First, all itemsets that have support above the minimum support, called *large itemsets*, are generated. In this case, the large itemsets are $\{1\}, \{2\}, \{3\}, \{1, 3\}$. Then, for each large itemset X, an association rule $X - Y \Rightarrow Y (Y \subset X)$ is derived if $support(X)/support(X - Y) \geq$ *minimum_confidence*. The results are 1 ⇒ 3 $(support = 60\%, confidence = 75\%)$ and 3 ⇒ 1 $(support = 60\%, confidence = 100\%)$.

The most well known algorithm for association rule mining is the Apriori algorithm[1, 2]. We have studied several parallel algorithms for mining association rules[14] based on Apriori. One of these algorithms, called HPA (Hash Partitioned Apriori), is discussed here. Apriori first generates candidate itemsets and then scans the transaction database to determine whether each of the candidates satisfies the user specified minimum support and minimum confidence. Using these results, the next candidate itemsets are generated. This continues until no itemset satisfies the minimum support and confidence.

The most naive parallelization of Apriori would copy the candidates over all the processing node and make each processing node scan the transaction database in parallel. Although this works fine when the number of candidates is small enough to fit in the local memory of a single processing node, memory space utilization efficiency of this method is very poor. For large scale data mining, the storage required for the candidates exceeds the available memory space of a processing node. This causes memory overflow which results in significant performance degradation due to an excessive amount of extra I/Os.

HPA partitions the candidate itemsets among the processing nodes using a hash function as in the parallel hash join, which eliminates broadcasting of all the transaction data and can reduce the comparison workload significantly. Hence, HPA works much better than the naive parallelization for large scale data mining. The k-th iteration (pass k) of the algorithm is as follows:

1. Generate the candidate itemsets:

   Each processing node generates new candidate itemsets from the large itemsets of the last ($(k-1)$-th) iteration. Each of the former itemsets contains k items, while each of the latter itemsets contains $(k-1)$ items. They are called k-itemsets and $(k-1)$-itemsets, respectively. The processing node applies the hash function to each of the candidates to determine the destination node ID. If the candidate is for the processing node itself, it is inserted into the hash table, otherwise it is discarded.

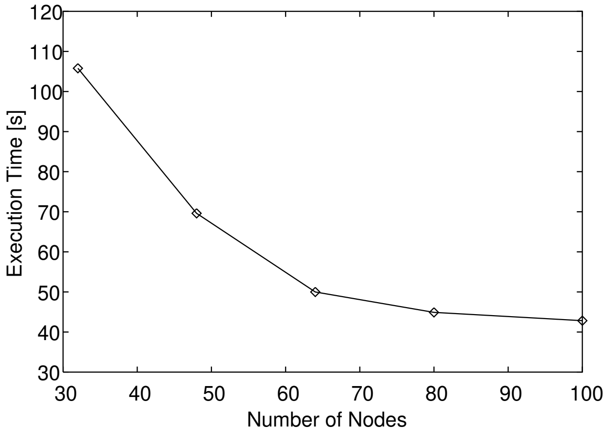

Figure 13: Execution time of HPA program (pass 2) on PC cluster

2. Scan the transaction database and count the support count:

   Each processing node reads the transaction database from its local disk. k-itemsets are generated from that transaction and the same hash function used in phase 1 is applied to each of them. Each of the k-itemsets is sent to certain processing node according the hash value. For the itemsets received from the other nodes and those locally generated whose ID equals the node's ID, the hash table is searched. If hit, its support count value is incremented.

3. Determine the large itemset:

   After reading all the transaction data, each processing node can individually determine whether each candidate k-itemset satisfy user-specified minimum support or not. Each processing node sends large k-itemsets to the coordinator, where all the large k-itemsets are gathered.

4. Check the terminal condition:

   If the large k-itemsets are empty, the algorithm terminates. Otherwise, the coordinator broadcasts large k-itemsets to all the processing nodes and the algorithm enters the next iteration.

4.2 Performance evaluation of HPA algorithm

The HPA program explained above is implemented on our PC cluster. Each node of the cluster has a transaction data file on its own hard disk. Transaction data is produced using data generation program developed by Agrawal, designating some parameters such as the number of transaction, the number of different items, and so on. The produced data is divided by the number of nodes, and copied to each node's hard disk.

The parameters used in the evaluation is as follows: The number of transaction is 5,000,000, the number of different items is 5000 and minimum support is 0.7%. The size of the data is about 400MBytes in total. The message block size is set to be 16KBytes, according to the results of communication characteristics of PC clusters discussed in previous section. The disk I/O block size is 64KBytes, which seems to be most suitable value for the system.

Note that the number of candidate itemset in pass 2 is substantially larger than for the other passes, which relatively frequently occurs in association rules mining. Therefore, we have been careful to parallelize the program effectively, especially in pass 2, so that unnecessary itemsets to count should not be generated.

The execution time of the HPA program (pass 2) is shown in figure 13 as the number of PCs is changed. The maximum number of PCs used in this evaluation is 100. Reasonably good speedup is achieved in this application as the number of PCs is increased.

5 Conclusion

In this paper, we presented performance evaluation of parallel database processing on an ATM connected 100 node PC cluster system. The latest PCs enabled us to obtain over 110Mbps throughput in point-to-point communication on a 155Mbps ATM network, even with the so-called ‘‘heavy’’ TCP/IP. This greatly helped in developing the system in a short period, since we were absorbed in fixing many other problems.

Massively parallel computers now tend to be used in business applications as well as the conventional scientific computation. Two major business applications, decision support query processing and data mining were picked up and executed on the PC cluster. The query processing environment was built using the results of our previous research, the super database computer (SDC) project. Performace evaluation results with a query of the standard TPC-D benchmark showed that our system achieved superior performance, especially when transposed file organization was employed.

As for data mining, we developed a parallel algorithm for mining association rules, and implemented it on the PC cluster. By utilizing aggregate memory of the system efficiently, the system showed good speedup characteristics as the number of nodes increased.

The good price/performance ratio makes PC clusters very attractive and promising for parallel database processing applications. All these facts support the effectiveness of the commodity PC based massively parallel database servers.

Acknowledgment

This project is supported by NEDO (New Energy and Industrial Technology Development Organization in Japan). Hitachi Ltd. technically helped us extensively for ATM related issues.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 207--216, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of International Conference on Very Large Data Bases*, 1994.

[3] A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, and D. A. Patterson. High-performance sorting on Networks of Workstations. In *Proceedings of International Conference on Management of Data*, pages 243--254, 1997.

[4] D.S. Batory. On searching transposed files. *ACM TODS*, 4(4), 1979.

[5] P.A. Boncz, W. Quak, and M.L. Kersten. Monet and its geographical extensions: A novel approach to high performance GIS processing. In *Proceedings of International Conference on Extending Database Technology*, pages 147--166, 1996.

[6] R. Carter and J. Laroco. Commodity clusters: Performance comparison between PC's and workstations. In *Proceedings of IEEE International Symposium on High Performance Distributed Computing*, pages 292--304, 1995.

[7] D.J. DeWitt and J. Gray. Parallel database systems : The future of high performance database systems. *Communications of the ACM*, 35(6):85--98, 1992.

[8] J. Gray, editor. *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann Publishers, 2nd edition, 1993.

[9] J. Heinanen. Multiprotocol encapsulation over ATM adaptation layer 5. Technical Report RFC1483, 1993.

[10] M. Kitsuregawa, M. Nakano, and M. Takagi. Query execution for large relations on Functional Disk System. In *Proceedings of International Conference on Data Engineering*, 5th, pages 159--167. IEEE, 1989.

[11] M. Kitsuregawa and Y. Ogawa. Bucket Spreading Parallel Hash:A new parallel hash join method with robustness for data skew in Super Database Computer (SDC). In *Proceedings of International Conference on Very Large Data Bases*, 16th, pages 210--221, 1990.

[12] M. Laubach. Classical IP and ARP over ATM. Technical Report RFC1577, 1994.

[13] D.A. Schneider and D.J. DeWitt. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In *Proceedings of International Conference on Very Large Data Bases*, 16th, pages 469--480, 1990.

[14] T. Shintani and M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proceedings of IEEE International Conference on Parallel and Distributed Information Systems*, pages 19--30, 1996.

[15] T. Sterling, D. Saverese, D.J. Becker, B. Fryxell, and K. Olson. Communication overhead for space science applications on the Beowulf parallel workstaion. In *Proceedings of International Symposium on High Performance Distributed Computing*, pages 23--30, 1995.

[16] T. Tamura, M. Nakamura, M. Kitsuregawa, and Y. Ogawa. Implementation and performance evaluation of the parallel relational database server SDC-II. In *Proceedings of International Conference on Parallel Processing*, 25th, pages I--212--I--221, 1996.

[17] TPC. TPC Benchmark^{TM} D (Decision Support). Standard Specification Revision 1.1, Transaction Processing Performance Council, 1995.