PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 47, NO. 1-2, PP. 5-36 (2003)

UTILISING NETWORKED WORKSTATIONS TO ACCELERATE DATABASE QUERIES

Mohammed ALHADDAD and Martin COLLEY

Department of Computer Science
University of Essex
Colchester CO4 3SQ, UK
e-mail: mjalha@essex.ac.uk, martin@essex.ac.uk

Received: Sept. 5, 2003

Abstract

The rapid growth in the size of databases and the advances made in Query Languages has resulted in increased SQL query complexity submitted by users, which in turn slows down the speed of information retrieval from the database. The future of high performance database systems lies in parallelism. Commercial vendors' database systems have introduced solutions but these have proved to be extremely expensive.

This paper investagetes how networked resources such as workstations can be utilised by using Parallel Virtual Machine (PVM) to Optimise Database Query Execution. An investigation and experiments of the scalability of the PVM are conducted. PVM is used to implement palallelism in two separate ways:

(i) Removes the work load for deriving and maintaining rules from the data server for Semantic Query Optimisation, therefore clears the way for more widespread use of SQO in databases [16, 5].

(ii) Answers users queries by a proposed Parallel Query Algorithm PQA which works over a network of workstations, coupled with a sequential Database Management System DBMS called PostgreSql on the prototype called Expandable Server Architecture ESA [11, 12, 21, 13].

Experiments have been conducted to tackle the problems of Parallel and Distributed systems such as task scheduling, load balance and fault tolerance.

Keywords: PQA Parallel Query Algorithm, PVM Parallel Virtual Machine.

1. Introduction

Exploiting idle workstations has attracted researchers, due to the fact that large portions of the workstations are unused for a large time and to the rapid growth in the power of workstations. It has been observed that, up to 80% of workstations are idle depending on the time of the day [8]. Commercially available Parallel Processing servers are expensive systems and do not present a viable solution for small-size businesses, therefore we are interested in trying to find alternative parallel processing methods and query optimisation methods. Such methods as described in this report utilise a network of workstations.

The goal of this research is to utilise any available computers in a data server's local network to optimise database query processing. Parallel Virtual Machine

(PVM) is a software that allows utilisation of networked workstations as a single computational resource. The effective use of PVM in enhancing the performance of Database Queries is presented. Experiments have been carried out and suggested that the task performed on the cluster of networked workstations are almost from 2 to 12 times faster than one workstation as explained in detail in Section 2.

The project goal was pursued in two separate ways as in *Fig. 1*:

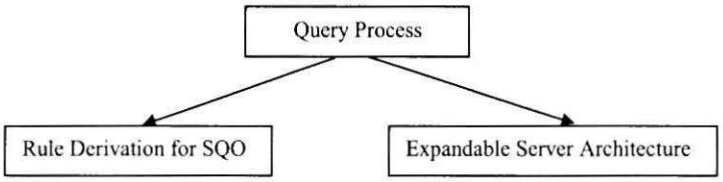


Fig. 1. the goal of the research

It is envisaged that these two separate ways to optimise query answering can be combined in an operational system, in which one workstation receives clients queries and chooses to answer each query either by Semantic Query Optimisation (SQO) and the original data server, or else by using the cluster of Expandable Server Architecture (ESA) machines.

The significance of the 'Rule Derivation for SQO' component of our research is that it greatly improves the applicability of semantic query optimisation techniques. The main objective of SQO is to use semantic knowledge (this knowledge has been represented in a form called a 'rule') to transform an original query into an alternative query that produces the same answer set but will be processed more efficiently by the data server and with lower-execution cost. In addition to the importance of learning rules automatically and using the derived rules for query optimisation, these rules also need to be maintained to keep the rules set accurate if the database can change. Therefore SQO becomes complex because the workload to derive and maintain rules can cancel the benefits of faster query processing.

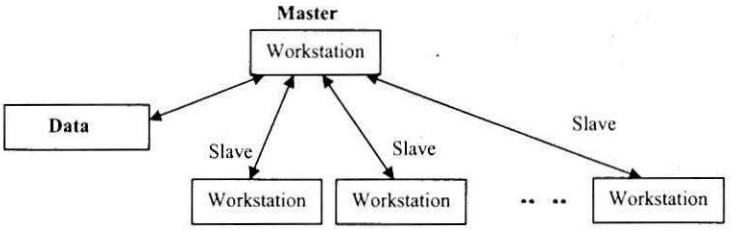


Fig. 2. Utilising Workstations for Semantic Query Optimisation

The demonstration of the performance of the systematic rule set derivation algorithm, which utilises multiple workstations, removes this problem, see *Fig. 2.*

This clears the way for more widespread use of SQO in databases, the detail is shown in Section 3.

The proposed 'Expandable Server Architecture' (ESA) allows the data server to spread from its original one workstation onto any other available computers in the local network. The effect is to create a distributed database within that network, and so gain the benefits of parallel processing, as described in Section 4. The set of general-purpose computers in this Expanded Server Architecture can be used as a separate data server from the original server from which the data was obtained. Therefore two queries can now be *simultaneously* processed: one on the original data server, and the other on the ESA cluster of workstations server. This latter server executes its queries using the proposed *parallel Query algorithm* PQA. The fault-tolerance problem has been tackled, if one of those servers is crashed, the current executed query will switch to the other server.

In order to demonstrate the proposed ESA idea, a prototype system has been built and experiments performed to measure execution times. A standard set of database tables has been used and a standard collection of SQL queries, in order to represent a realistic query processing environment. The TPC-H standard database benchmark provided the data and the queries [10]. The authors do not claim to draw any conclusions about performance on an actual TPC-H benchmark. This database schema consists of 8 tables and was distributed statically into a cluster of 8 workstations in the experiments. These 8 workstations are the upper limit that have been provided for this research.

The proposed Parallel Query Algorithm (PQA) works as an Interface Manager over the ESA, which receives the users queries and decomposes them into sub-queries as described in Section 4. In special cases where the sub-query has an error from an early termination of the query execution, an error message is returned to the user. The Query Processing Algorithm is developed by employing both inter- and intra-operation parallelism. The proposed algorithm is able to perform adaptively based on two methods: the Dynamic Rescheduling Method where the processors are allocated to tasks during runtime on the fly, and the Merge-Join Method. There are two main factors that influence processor assignment: communication time and load balancing [7]. Communication costs consist of the data transmission costs and the overheads for co-ordinating multiple processors; it is an important component of the total cost depending on the network environment and the database placement. On the other hand, load balancing tremendously influences overall performance because the overall query execution time or the individual phase execution time is determined by the longest execution time over multiple processors. The experiments in Section 5 represent the performance of the algorithm on only a single data set and a few specific queries.

A huge amount of CPU and memory resources are required in order to efficiently process distributed N-way join queries on huge data sets. Therefore, one main objective of this architecture is to utilize the computer resources of the clustered networked workstations to meet this demand. Thus, *Parallel Query Algorithm* PQA [11] is implemented on client-server architecture with a configuration, where a master process running at the query initiated site which manages a virtual pool of

lightly loaded slave workstations. Each slave workstation can dynamically join and quit the pool, depending on its participating to answer the original query. At any moment, the computing power of the virtual pool can be fully utilised to process the original query. The master and slaves are interconnected via a fast local area network.

The proposed '*Expandable Server Architecture*' ESA allows the data server to spread from its original one workstation onto any other available computer in the local network by using *Parallel Virtual Machine* PVM [12]. The effect is to create a distributed database within that network, and so gain the benefits of parallel processing. The set of general-purpose computers in this ESA can be used as a separate data server from the original server from which the data was obtained. Therefore two queries can be *simultaneously* processed now: one on the original data server, and the other on the ESA cluster of workstations server. This latter server executes its queries using the proposed PQA. The fault tolerance problem has been tackled; if one of those servers is crashed, the current executed query will switch to the other server.

The structure of the paper is as follows. In Section 2, an investigation of using PVM to create a cluster of workstation is conducted. Section 3 explains utilising cluster of networked workstations to create a rule set for *Semantic Query Optimisation*. An overview of PQA, Data Placement, Dynamic Schedule Allocation and fault tolerance in PQA are given in Section 4. The results of practical experiment and method to measure the response time are demonstrated in Section 5. Finally, Section 6 is the conclusion.

2. Parallel Virtual Machine PVM

PVM is a software system that allows the combination of a number of computers, which are connected over a network into a parallel virtual machine. This machine can consist of computers with different architectures, running different operating systems and can still be treated as if it were a single parallel machine. As the software is public domain this means that many organisations which already have a network of workstations can get a parallel machine for free and solve larger problems using existing hardware resources. PVM is a small package about 1 Mbyte and easy to install. It needs to be installed once in all machines that are desired to form the virtual machine. PVM system uses the message-passing model. In this, sets of processes are invoked. Each process has its own local memory. Processes communicate by sending and receiving messages, and thus the transfer of data between processes requires co-operative operations to be performed by each process (a send operation must have a matching receive).

PVM communication model assumes that any task can send a message to any other PVM task and that there is no limit to the size or number of such messages. While all hosts have physical memory limitations that limits potential buffer space, the communication model does not restrict itself to a particular machine's limitations

and assumes sufficient memory is available.

The PVM communication model provides asynchronous blocking send, asynchronous blocking receive, and non-blocking receive functions. A blocking send returns as soon as the send buffer is free for reuse, and an asynchronous send does not depend on the receiver calling a matching receive before the send can return. There are options in PVM 3 that request that data be transferred directly from task to task. In this case, if the message is large, the sender may block until the receiver has called a matching receive. A non-blocking receive immediately returns with either the data or a flag that the data has not arrived, while a blocking receive returns only when the data is in the receive buffer.

2.1. PVM Scalability

Some database tables are much too large to be distributed to ordinary workstations, because the local storage capacity on these general-purpose computers is not large enough to accommodate the database tables. Therefore, there is an upper size limit for tables, beyond which the data distribution approach is not applicable. Performance also declines with increasing table size, before that upper size limit is reached. The time taken to send data from the master workstation to a set of slave computers was investigated. Tables of progressively increasing size (from 32560 to 846585 rows, representing database tables up to 93 Mbytes) were sent to sets of 1, 2, 3 ... 8 workstations and the total send time measured. The following graph displays the results. They show that even these relatively small tables suffer from performance degradation related to their size.

Each table size is shown as a curve on the graph. Small tables appear as horizontal lines near the bottom of the graph. Curves are seen to deviate progressively more from the horizontal as the table size increases, but all become approximately horizontal when the number of hosts becomes 'sufficiently large'. Using more hosts reduces the size of the data set that is sent to each computer, because the number of hosts divides the table. The graph reveals that above a particular data size per computer the time to transfer data between *Master* and *slaves* increases dramatically. All curves become horizontal when the number of table rows per host is 150 000 or less. So 150 000 rows for this 112-bytes-per-row table is the maximum size per host (for these particular hosts) to avoid the delay. 150 000·112 bytes = 16 Mbytes. For FAST operation, the maximum table size is 16·H Mbytes, where H is the number of workstations available for use. Larger tables can be processed, but time will increase significantly because of the data transfer time component shown in the graph.

Paging in the Receive Buffer memory space in the slave workstations causes the large increase in data transfer time above 16 Mbytes per workstation. The next physical limitation as table size increases beyond 16·H Mbytes is the size of the swap file used for page-swapping, since our system operates in the virtual memory of the workstations. The size of the swap file can be increased up to the limitation of

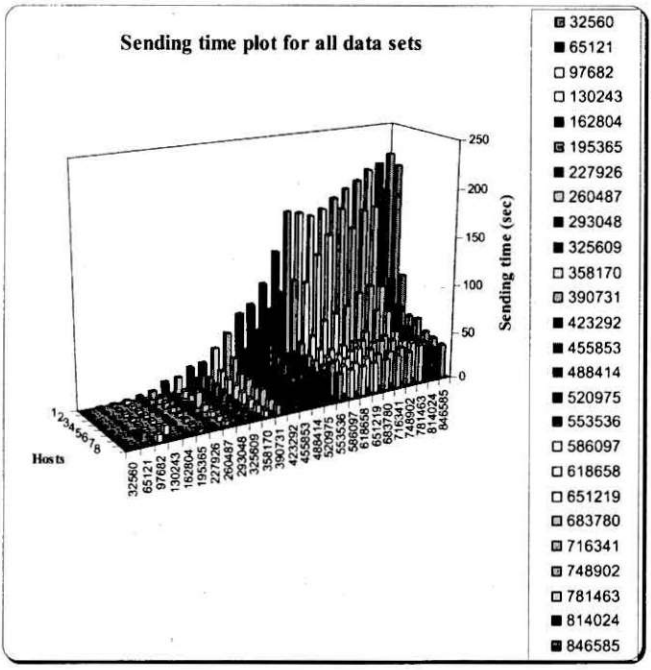


Fig. 3. Sending different size of data set over a cluster of 8 hosts

the available disk space accessible to each workstation. The swap file can be placed on any mounted drive, but a computer can slow down dramatically if a remote (shared) disk is used for virtual memory swap space. A large network accessible disk increases the maximum size of database tables that can be processed, but the resulting slowdown of the workstation (which affects all programs running on it, not just our background processes) is a clear drawback. The workstations used in the experimental network are chosen as typical examples of ordinary PCs in current use, not state-of-the-art machines. Their internal disks are of 10 Gbytes capacity. They have Intel PIII 450 MHz processors, 128 Mbytes of main memory, Windows NT or LINUX operating system, and communicate by Fast Ethernet.

2.2. *Virtual Machine vs Local Memory*

The term Virtual Machine is used to refer to a logical distributed-memory computer. People usually run tasks not bigger than the physical memory due to the performance gap between the processor and the disk by using the virtual memory mechanism. Some of them prefer to buy more DRAM to fit the task. Or they might get a bigger computer to hold the task. Virtual machines solve this problem. The high-bandwidth network, fast network and PVM system can utilise all the resources

efficiently. *Fig. 4*, shows the comparison of performance between the *virtual machine system* and the *single system* for sending database table of various sizes. We can see that the *virtual machine system* seems to perform better than the *single system* for all sizes of database tables.

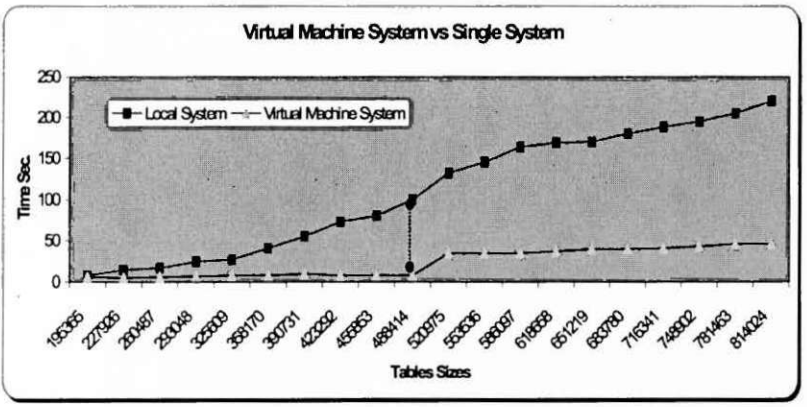


Fig. 4. Calculated elapsed time of different data set sent over a Virtual Machine System and a Single System

To quantify the difference in performance between sending different database (tables) to *single system* (one workstation) and to *virtual machine system* (cluster of 8 workstations) as shown in *Fig. 5a–b*, we can fit the two lines on the graph using linear regression models and compare the estimated coefficients:

$$\text{Single} = -80.339 + 0.000378x$$
$$\text{Network} = -17.492 + 0.00008142x,$$

where x is the size of Database tables.

However, inspectig the *virtual machine system* line in *Fig. 4*, we can see a pronounced 'jump' at a database size of around 488414. This is due to the swap paging

mechanism that comes into effect at this point. Therefore it seems reasonable to compare separately the performance *before* and *after* this takes place.

In this case, comparing the two coefficients, the gradient tells us that as we increase the database size, time increases at a rate 4.6 times slower for virtual machine system in comparison to single system ($0.000378/0.00008142 = 4.6$).

Table 1 tells us, on average (over all the database sizes), that the virtual machine system ($M = 23.6$, $N = 20$, $SD = 16.9$) performs about 5 times faster than the single system ($M = 110.45$, $N = 20$, $SD = 73.765$). The virtual machine system is always at least 1.3 times faster, in fact, it can be up to 12 times faster than

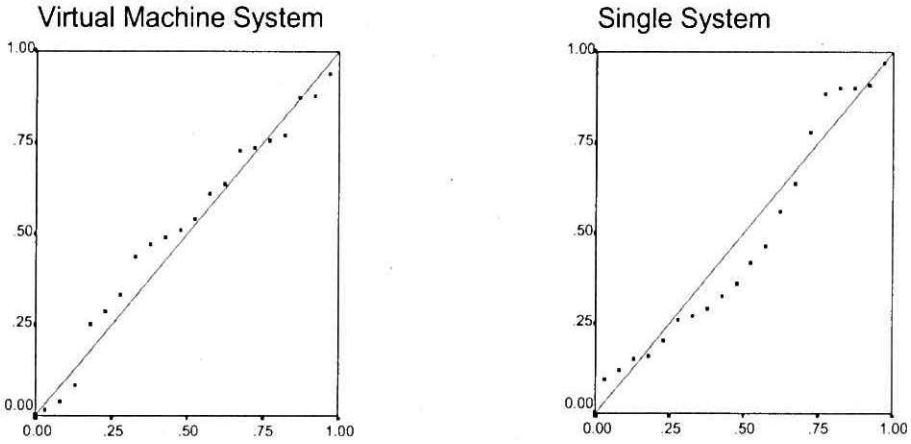


Fig. 5. (a) Fitted for the Network RAM, (b) Fitted for the Single RAM

the single system. The performance of the two systems is significantly different from each other at 0.001 level.

2.3. What is the Optimum Number of Workstations to Handle a Certain Size of DB

In conclusion, one can say that by expanding the network with more processors the size of the database that can be handled in an optimum way, i.e with time ratio near the optimum, will increase.

Furthermore, as the decay in the performance of the network is due to excess page-swapping, it is reasonable to assume that increasing the RAM on each node will again lead to larger databases being optimally handled by the network. This is limited to the maximum capacity of RAM in each node.

Graph *Fig. 6* shows the number of workstations against the database size measured in units of 32561 rows of data (about 3.48 MB). The data points give the optimal number of workstations required to handle databases of increasing size. These points are taken from the previous experiment see Section 2.1.

A line of best fit through the data points can be extrapolated to predict the optimal number of workstations required to handle databases of any given size. For example, a multiple of 60·(32561) rows of data will require around 15 workstations to be handled optimally.

3. Utilising Network Resources for Speeding up the Query Process

Query Optimisation is a part of the relational DBMS with speeding up queries

Table 1. Ratio of Virtual Machine System over Single System

Database Sizes	Single System	Virtual Machine System	Ratio of Network over Single
195365	8	6	1.333333333
227926	15	5	3
260487	17	6	2.833333333
293048	25	7	3.571428571
325609	27	8	3.375
358170	41	8	5.125
390731	55	10	5.5
423292	73	8	9.125
455853	80	8	10
488414	100	8	12.5
520975	132	35	3.771428571
553536	145	35	4.142857143
586097	164	35	4.685714286
618658	169	37	4.567567568
651219	170	40	4.25
683780	180	40	4.5
716341	188	41	4.585365854
748902	195	43	4.534883721
781463	205	46	4.456521739
814024	220	46	4.782608696
Mean	110.45	Mean 23.6	Average 5.032002141
SD	73.7645	SD 16.9	max 12.5
			min 1.333333333

executions as the main goal. There are three main optimisation approaches to improve the query processing: algebraic/graph-based, systematic, and *Semantic Query Optimisation*. The authors focus on the third approach during this research.

3.1. Obstacles in Semantic Query Optimisation Approach

Semantic query optimisation uses semantic rules to transform a given query into alternatives, and then selects the optimum query between the alternatives according to their cost. These alternative queries can be different syntactically but must be the same semantically.

There are various techniques for Semantic Query Optimisation. To the best of

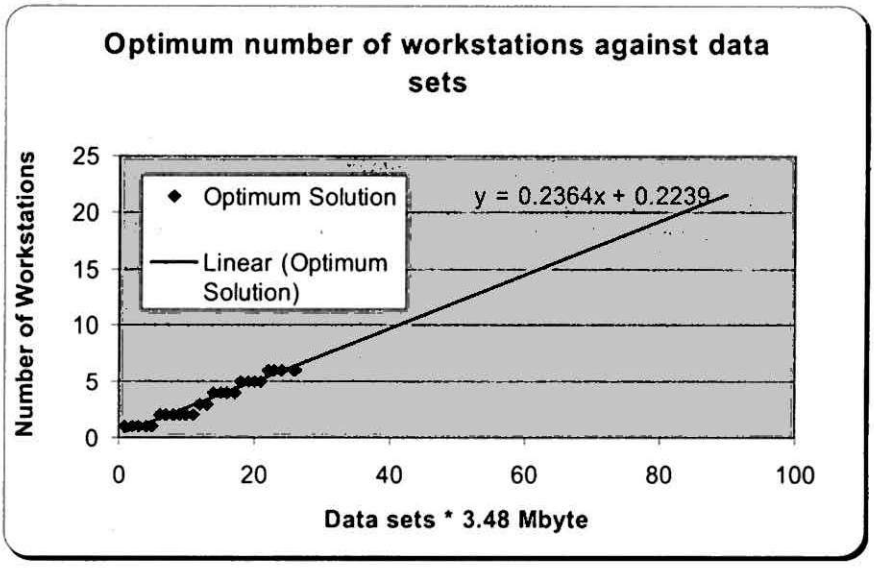


Fig. 6. Optimum Numbers of Workstations against DB Size

the authors' knowledge, there are no broad commercial implementations of SQO. There are numbers of reasons for this lack of implementation. First and foremost, because SQO has been associated for many years with cases designed for deductive databases, it was not thought be useful for other uses such as relational database technology [15]. Second, it has been commonly assumed that for an SQO to be of benefit, many integrity constraints have to be defined for a given database. Otherwise, queries could not be optimised semantically. Finally, the speed of the CPU and the I/O at the time when the Semantic Query Optimisation was developed was different than nowadays. SHEKER in his paper [17] shows that the cost of semantic optimisation could be comparable to the query execution cost.

Semantic Query Optimisation approach can use the previous queries to improve the future queries. Therefore, the first query will be executed straight away because no rules exist in the rule set. The conditions in the second query will add up into rule derivation process, then to rules set. In other words, the system builds its own rules, thus it would answer a certain query. Moreover, this rule would be useless when any database changes.

In this paper, the authors used the attribute pair rule [16]. These rules are also created automatically using *QuickSort* and *Scan Bucket Algorithm* for semantic query optimization. Due to the space restriction, readers can see the details in [16].

3.2. *Sorting Data Subsets for Rules Set Derivation and Maintenance*

The data in a Database table or view is partitioned by the *M*aster workstation whatever number of workstations is available. For an N-row table and H workstations, each *Slave* workstation receives N/H rows. Partitioning is done by counting rows rather than examining data values, so it is fast. The *M*aster workstation also tells the *Slaves* which attribute to use as antecedent for the current rule set. Each slave then sorts its sub-table on that attribute, extracts a rule set from the sorted data, and sends the rule set to the *Master* workstation. The *Master* merges the sets of rules, one from each slave, into a single set for that antecedent attribute. Receiving and merging rule sets is much faster than merging data sets, because rule sets are small (e.g. 100 rules per set). It takes less than one second to receive and merge rule sets derived from a 400 000-row table, for example, for up to ten *Slaves*.

When the antecedent attribute is *numeric*, the master tells the *S*laves how many rules to derive. The *M*aster also broadcasts the MIN and MAX values for the attribute so that all *S*laves use the same set of sub-ranges as rule antecedents. The number of rules produced per slave is therefore constant for numeric antecedents. Sorting the data makes it easy to extract a histogram rule set since the antecedent attribute values are all arranged in order. It makes also *rule maintenance* easy.

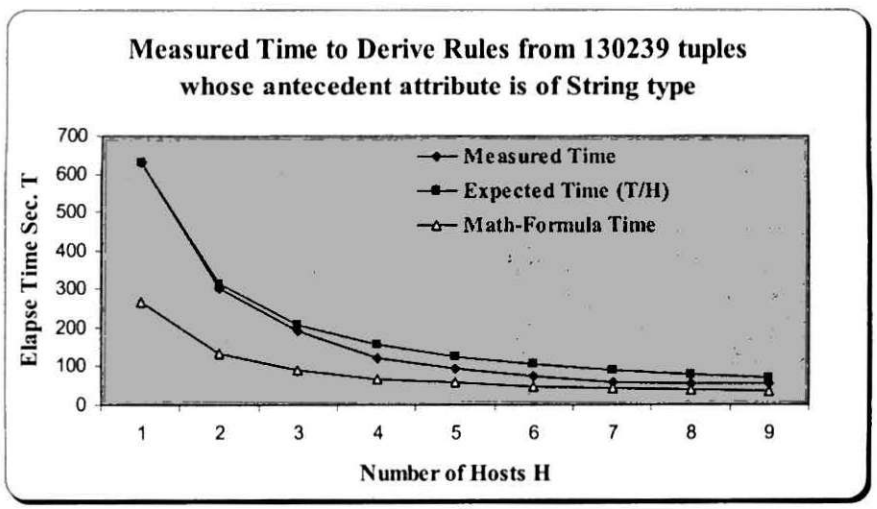


Fig. 7. Measured time for rule set derivation shows better than linear speedup

No. of Hosts, H	1	2	3	4	5	6	7	8	9
Measured time (sec)	625	300	190	120	94	75	57	56	55
Expected $625/H$ (sec)	625	313	208	156	125	104	89	78	70
Math-Formula (sec)	267	134	91	68	57	49	43	39	35

Measured time means the observed time taken to create a set of rules (a histogram rule set) from the 130239-row database table where each row is of 81 bytes. The roughly hyperbolic curve for measured time suggests xy = constant. In this case we might expect the constant to be 625 seconds, the measured time for one workstation; time to complete a task being inversely proportional to the number of workers involved. *Expected Time* in the graph is therefore calculated as $625/H$ where H is the number of workstations used in the local network. The measured experimental results show a close correlation to the predicted results, as indicated in *Fig. 7*. As expected, the measured results are better than the expected results. This is due to the 'fixed ratio' used by the estimation function, whereas the actual measured time decreases by a 'variable ratio'. The explanation for this better than expected predicted performance is partly the computational complexity of the *QuickSort* algorithm used. It has a best case complexity of $N \log_2 N$ for N data items, and a worst case of N^2. We divide N by H and sort the smaller subsets, without the need to recombine the sorted subsets (only the relatively small sets of *rules* are merged). A second factor in the speedup is the amount of virtual memory paging involved. Each page swap between disk and main memory is a significant time delay. Large datasets do proportionately more page swapping. The third trace shows the *Mathematical Formula Time* which we calculate by summing the *Computation Time* with the *Communication Time* as can be seen in *Table 2*. Computation Time $= N + (N/H) \cdot \log_2(N/H)$ average processing time for a single operation.

Computation Time = number of words to send over the network· time to send 1 word. It is a simple phenomenological model to calculate the execution time which does not take into account the factors listed for the *Measured Time*.

Table 2. Results for the Mathematical Formula Time Model

H	N/H	$N + (N/H) \cdot \log_2(N/H)$	Communication	Math_Formula Time
1	10549359	256452361.3	10.55	267.00
2	5274679.5	128336389.6	5.27	133.61
3	3516453	87017052.99	3.52	90.53
4	2637339.75	65424431.49	2.64	68.06
5	2109871.8	54875072.49	2.11	56.98
6	1758226.5	47024979.52	1.76	48.78
7	1507051.29	41479019.89	1.51	42.99
8	1318669.88	37358777.05	1.32	38.68
9	1172151	34180774.97	1.17	35.35

Fig. 7 shows that when 9 workstations were used, it took only 55 seconds to distribute and sort the data sub-sets, derive 9 separate histogram rule sets and merge them into a single rule set in the master workstation. The same operation performed on a single workstation is seen to take over 5 minutes. The practical significance of

this acceleration is that rules can now be generated in response to a query and may be available in time to be used to optimise the next query. This *query-triggering* of rule set derivation is now a feasible alternative to *speculative generation* of sets of rules from database tables before queries arrive. The experiments were repeated with various database tables. They varied in antecedent type, table size, degree of prior sorting and total number of workstations used. The graph above is representative of the results to some extent, but larger database tables needed correspondingly larger numbers of workstations to provide fast derivation. Furthermore, the minimum time achievable increased with the size of the database table because data are sent to computers before they start work on it. Data subsets must be sent one after another on the local network until the whole table has been distributed. The network bandwidth therefore imposes a time proportional to table size on the whole process. (Some workstations will have finished their tasks before the final data subset is sent). This undesirable time penalty can be removed by distributing database tables to workstations in advance. Then rule sets can be derived in a few seconds by broadcasting only the *derivation parameters*. These include the identity of the antecedent attribute, and if it is a numeric attribute, the number of rules required in the set plus the MIN and MAX values in that column of the whole table.

The master broadcasts to the slaves all data changes. The slaves then revise their rules and notify the master of any changes. The master obtains an updated rule set describing the changed database table in less than 2 seconds by this method. Sorting data is usually a slow operation. This would be a disadvantage for the current application, because rules are needed for query optimisation as soon as possible after a query reveals user interest in certain columns of some virtual or base relation. If rules are not produced until the data is sorted, then sorting must be done as fast as possible. Our experiments show sorting is significantly accelerated by the parallelism in distributing data to multiple workstations.

3.3. Scan Bucket Algorithm to Derive Rules for SQO

Rules can also be derived using a more direct algorithm, which scans once through the database table. During the scan each tuple is mapped to a bucket in a set of buckets corresponding to the required set of rules. Buckets correspond to bars in the histogram.

For numeric antecedent attributes, the number of bars and their sub-range limits are known in advance. So mapping each tuple to its bucket is achieved by matching its value for the antecedent attribute to the relevant sub-range. For string antecedents a new bucket is added for each new value of the attribute encountered during the scan. Each bucket has one rule associated with it, which describes all tuples mapped to that bucket so far. The subset descriptor evolves as more tuples are added to the bucket's subset. *For example*, at some point in the scan one subset descriptor has the form:

$$(15 \leq a \leq 30) \Rightarrow c(71 \leq c \leq 94) \wedge (101 \leq g \leq 156)$$

This indicates that all tuples encountered so far whose attribute 'a' value was in the range $15 \leq a \leq 30$ were found to have values of attribute 'c' in the range 71..94 and attribute 'g' values in 101..156. If the next tuple in the table has values $a = 16$, $c = 96$ and $g = 121$, then the value of the antecedent attribute 'a' maps it to the bucket with antecedent range ($15 \leq a \leq 30$). The value $c = 96$ requires the range in the assertion describing all 'c' values to increase from (71..94) to (71..96), and the value of 'g' does not change the descriptor because 121 agrees with the assertion that all 'g' values are in the range 101..156.

3.4. Measuring the Algorithms Performance

The scanning algorithm for rule set derivation has the advantage that a single pass through the data generates a rule set. This is much faster than sorting. The disadvantage is that sorted data, to support subsequent rule maintenance, are not produced. These measurements, as shown in *Fig. 8*, are for a 42 Mbyte table with 390731 rows. The scanning algorithm times form a horizontal line on the graph. Elapsed time was 30.5 ± 3.5 sec, independent of H. The time to derive the same set of rules by sorting the data subsets in the workstations varied from 4018 seconds for one workstation down to 205 seconds for six machines.

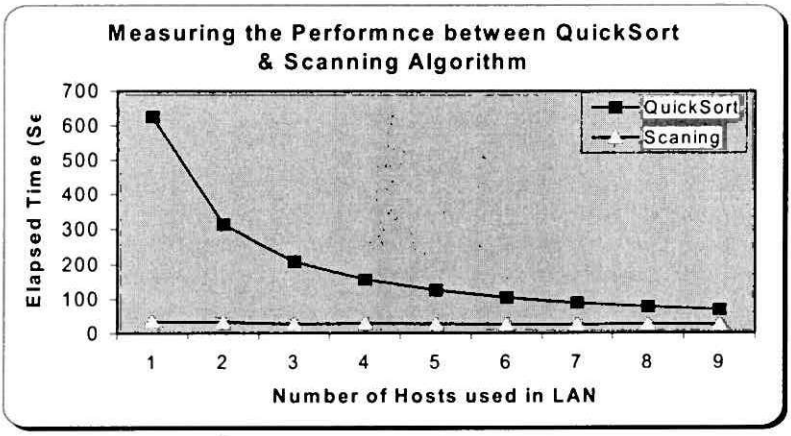


Fig. 8. Comparing rule set Derivation times for QuickSort & Scan Algorithm

Since the scan times are fairly constant, independent of number of processors, it is possible to derive *multiple rule sets* simultaneously, one in each host. Each rule set has a different antecedent attribute. The *whole* data table is now broadcast to N workstations so that the derivation time for N *sets* of rules is still about 30 seconds for this 42 Mbyte table.

Two or more *sets* of rules can be produced during the scan in a single computer. A set of buckets are provided for antecedent attribute 'a' and another set for

antecedent '*d*', say, in another rule set. Then in each tuple the value of attribute '*a*' maps it to a bucket in the first rule set, and the value of attribute '*d*' maps it to a bucket in the second evolving rule set.

3.5. Consistency of the Rules

The sorted data subsets in each workstation are useful for deriving rule. Moreover, it's useful for rule maintenance as well. It makes *rule maintenance* easy. The master broadcasts to the slaves all data changes. The slaves then revise their rules and notify the master of any changes. The master obtains an updated rule set describing the changed database table in less than 2 seconds by this method. For more details due to space restriction, please consult [5, 12].

4. Utilising Network Resources for Answering Query

Query processing is the crucial part of the DBMS, which is responsible for generating the best plan to execute the query. After receiving a query from the user, it has to be transformed to a relational algebra expression and then parser during the transformation. The next step is to generate *Access Plans.* From these plans the optimal one will be chosen, taking into consideration the methods of accessing this data and the physical feature of these data. In general, query processing involves the costs of processing *Input/Output* and communications.

Querying a database is the most important part of any database activities. The purpose of querying the database is not only to satisfy the query but to minimise the response time. Therefore, maintaining a reasonable level of performance is essential. The response time of a query (the time difference between the time the query arrives and is answered) is the sum of waiting time and execution time. The overall response time essentially can be reduced by:

- Reducing the average waiting time of a query: this refers to the time difference between when a query arrives and when it starts being executed.
- Reducing the execution time of a query: this refers to the time difference between the start and finish of the execution of a query

4.1. Expandable Server Architecture ESA

Applying parallel processing techniques, like Parallel Query Processing in database systems, may improve the Database Query answering time and hence the overall response time of a query. The need for this improvement has become apparent due to the increasing size of the relational database as well as the support of high-level query languages like SQL, which allows users to present complex queries.

Expandable Server Architecture (ESA) has been designed to accomplish that by utilising the resources of any Local Area Network (LAN) such as a small business. This means that we are making use of the workstations that are connected in the LAN and saving the small business from buying an expensive system. It is a special class of parallel processing systems, which falls in the category of distributed-memory architecture where a set of workstations are interconnected through a Local Area Network and they communicate with each other by sending messages across the interconnection network. Each workstation has its own private memory, disk, CPU and local communication (between disk, memory, and act) and has access to a global interconnection network.

However, using cluster of workstations for database query processing poses several problems and performance issues. As in NOW [18] there is no central control therefore it is impossible to distribute the database among workstations, the database relations are stored in central workstation. Like Parallel Database Systems, ESA with *Parallel Query Algorithm* PQA [12] has partially central control and the database is partitioned across the clustered workstations, this reduces overhead as the data does not need to be sent to the other workstations.

4.2. Parallel Query Algorithm PQA

A main function of Query Processing is to transform a high-level declarative query into an equivalent lower-level procedural query. The transformation must achieve both correctness and efficiency [14]. The well-defined mapping from relational calculus to relational algebra makes the transformation correct, efficient and easy, but producing an efficient execution strategy is more involved. The lower-level query actually implements the execution strategy for the query. Since each equivalent execution strategy can lead to a very different consumption of computer resources, the main difficulty is to select the execution strategy that minimises resources consumption.

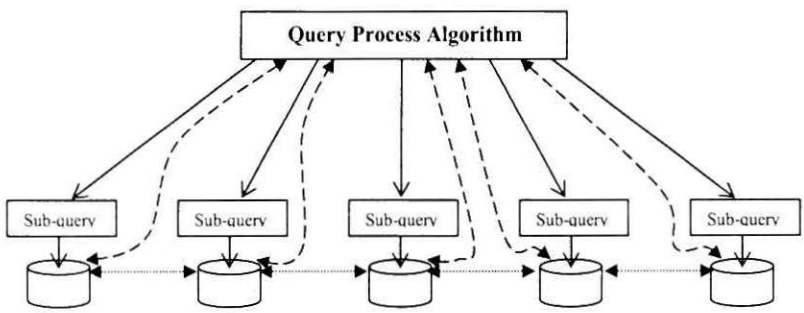


Fig. 9. The parallelism in ESA by PQA

The combination of parallel processing and the database management gave rise to the concept of Parallel Database. *Parallel Query Algorithm* PQA exploits the parallelism available in ESA to bring high performance database, see *Fig. 9*. In NOW [20] one workstation has control over the database and the others have access to the database through this workstation. As a query indicates which relations are involved and central database workstation transfers all required relations to the workstation initiating the query. PQA represents partially *central control*, therefore the database relations can be partitioned and distributed over the clustered workstations but deals with only read-query. By partially *central control* we meant that PQA has metadata of how the database scheme is partitioned, the size of each partition, data structure and location of the data partition. Due to the limited number of workstations that small business might use and to limited number of workstations that this study is dealing with, the non-query load (the back-ground load) is not considered. The other problem is the work load as this study is not a simulation study as in NOW, a real data about 2GB is being used from TCP-H and *Data Placement Algorithm* (see Section 4.4) is used to tune static data distribution among the cluster of workstations to achieve an optimal performance.

4.2.1. Parallel Query Algorithm Components

In this Section a description of PQA and all its components will be presented. *Fig. 9* shows all the components and processes, which are executed on all workstations at the same time. *Query Manager* is one of the components (it's an arbitrary processing node) of PQA. At the initialisation stage, this process receives user's query and reads the metadata, then it is responsible for undertaking the entire execution plan and keeping up a correspondence between all nodes. *Query Manager* has two sub-components (*Scheduler, Information Policy* and *Decompose Query*), these sub-components co-operate with each other in order to schedule dynamically query execution plan. *Information Policy* reads the information from the metadata. When a query arrives, *Decompose Query* will in turn divide up this query into sub-queries as can be seen in Section 4.2.2. *Scheduler* will receive these sub-queries and then allocate each sub-query to a node based on the knowledge received from the *Information Policy* by spawning a process (*Slaves*) in those nodes, every node has a unique processor identifier (PID) that is used by *Scheduler* and *Slave* to communicate with each other. When one of the *Slave* processes finishes its task (fetching data), it sends an acknowledgement to *Scheduler* with the structure of *Intermediate Relation* IR and its size, *Scheduler* passes this information to *Information Policy* to update its information, and makes a decision of the best optimal execution strategy for the next operation, either to send it to available *Slave* for joining operation or to sort IR based on the join attribute if its not sorted (detailed analysis of Dynamic Scheduler will be discussed in Section 4.3). *Slave* is the other component of PQA, it starts fetching data when it receives the sup-query from the *Scheduler*, then it sends an acknowledgment to *Scheduler* telling it that the task is finished. Decision

Table 3. Message Tags

Process	Tag Identifier	Tag	Description
Scheduler	sub-query	10	Send sub-query to *Slave* to start fetching data
	how many partition	19	Send to *Slave* how many partitions in table
	start join IR	14	Send the salve to start joining
	finish successfully	11	Send to *Slave* to exit
	Start sort IR	13	Send to *Slave* to start sorting IR
	Start send IR	18	Send to *Slave* to start send IR to peer *Slave*
	Start receive IR	17	Send to *Slave* to start receive IR from peer *Slave*
Slave	Finish joining	15	Send to *Scheduler*, join is finished
	Finish sorting	20	Send to *Scheduler*, sorting is finished
	Finish Concatenate	9	Send to *Scheduler*, finishing concatenate IR
	Final result	16	Send to *Scheduler*, final result
	Fetch sub-query	12	Send to *Scheduler*, finished fetching sub-query
	Finish receive IR	22	Send to *Scheduler*, finished receiving IR
	Finish send IR	21	Send to *Scheduler*, finished sending IR
	Commit_Sub-query	9	Finished fetching data.

is made by the *Scheduler* and sent to *Slave* telling it either to send IR to peer *Slave* or to receive IR from peer *Slave* or sort its IR according to join predicate. Since the behaviour of the components varies and they receive different acknowledgement messages, a table of message tags is maintained, see *Table 3*. In addition, for better understanding of the flow of interactions between processes, these message tags are shown in *Fig. 10*.

4.2.2. *Query Decomposition Algorithm*

The query must be broken up into sub-queries to run on the separate workstations. Two versions of this query decomposition process were investigated. *First*, to split the initial query into sub-queries and then let the *Join Manager* manage the sorting and joining of the intermediate relation results. *Second*; the decomposition routine tries to speed up the Join procedure by sorting the *Intermediate Relation*. This is done by using the advantage of having DBMS PGSQL installed in each site, which allows adding ORDER BY clause to the sub-queries. Due to space restriction, more details could be found in [5], [12], and [13].

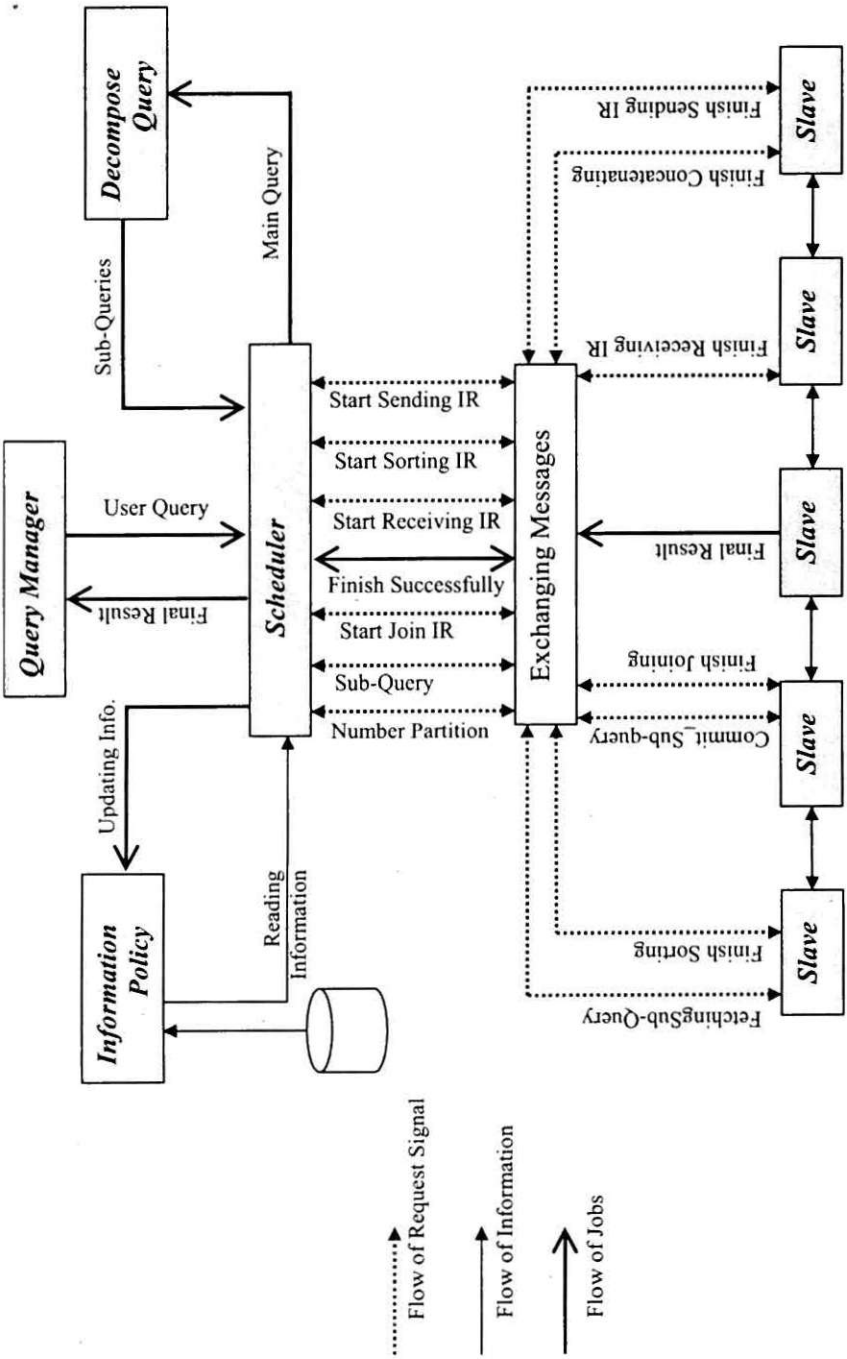


Fig. 10. Flow of Messages in PQA

4.3. Dynamic Load Scheduling

A query evaluation plan is generated by the optimiser. The task of the optimisation is broken into phases, for example, scheduling, algebraic transformation, etc. The decision of which step to apply next is based on cost estimations. Thus the quality of the optimisation result depends on the accurateness of the cost prediction. The problem that arises is how to predict the optimal cost. To solve this problem, many information parameters can be obtained during the query execution, this gives the accurate prediction needed. Consequently, pushing certain optimisation steps into the execution phase can alleviate the problem of optimisation in parallel database systems.

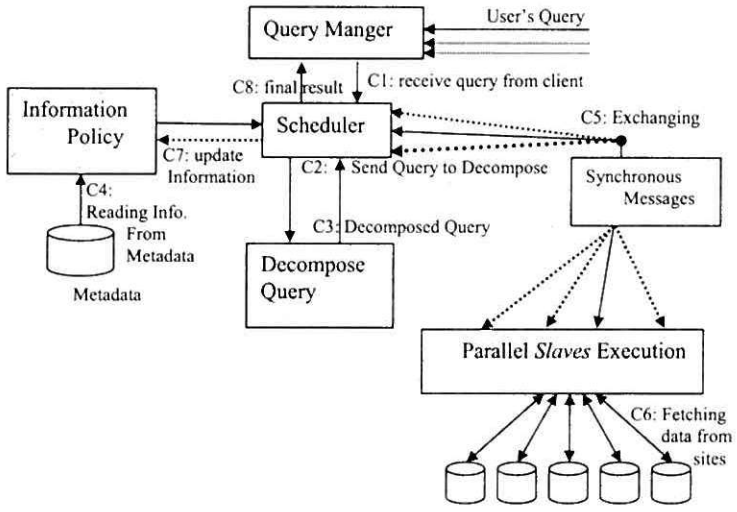


Fig. 11. Parallel Query Algorithm

In the proposed algorithm PQA, the query is received by *Query Manager* process and then decomposed into sub-queries by using the *Decomposition* algorithm. A process in the remote site in ESA handles retrieving the data by the algorithm called *Slave* see Appendix A and B for the algorithms *Slave* and *Query Manager* respectively.

When a new user's query arrives as shown in *Fig. 11*, an arbitrary processing node '*Query Manager*' receives it and becomes the co-ordinator in charge of optimising and supervising this query (C1) and passes it to *Scheduler*. The *Scheduler* first determines the degree of parallelism for the query by passing the main query to *Decomposing Query* (C2), it returns sub-queries (C3). (C4) determines the number of *Processing Sites* (PSs) and number of disks that hold the data partitions and passes it to *Information Policy*. Through exchanged messages between *Scheduler* and *Slave*, each operator can process the output of the previous one without delay, by sending knowledge to the *Scheduler* telling it that the task has been finished

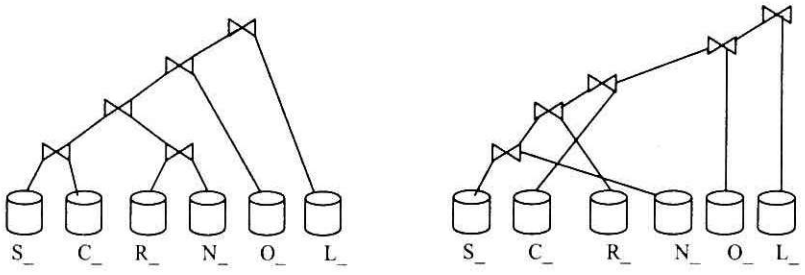
and it is ready for next task (C5). Slaves start fetching data when they receive the sub-query (C6). Accurate information such as size and structure of intermediate relation will be updated in (C7). The Query will be considered to be answered when the slave sends a final result to *Scheduler* (C8)

Message passing is used for transferring data and messages between the *Scheduler* and the *Slave* processes. An accurate description of the data is sent to the *Scheduler* such as the size of the intermediate result and which processes have finished their work. The *Scheduler*, in turn, dynamically allocates the next step. This step is either to send a command to slave process to sort their *IR* or to send a message to available *Slave* to receive *IR* from the peer slave for joining or concatenating.

All steps taken by the *slaves* are managed and controlled by the *Scheduler*. At a given moment, the *Scheduler* will order a *slave* to undertake a specific task. This is achieved by exchanging messages as shown in *Fig. 10*. The dynamic scheduling of tasks at run time begins when a *slave* receives the message 'FETCH SUBQUERY' to fetch a sub-query, then the Intermediate Relation is obtained, known as *IR*. Subsequently, the *slave* sends an acknowledgement message 'COMMIT_SUBQUERY' and the size of *IR* to *Scheduler* indicating that fetching has been completed. *Scheduler* may, on one hand, send a message 'SEND_IR' of sending data to one *slave* after commanding that *Slave* to sort IR according to *Join Rule*. On the other hand, it gives a separate order 'RECEIVE_IR' for receiving *IR* from a peer slave, taking into consideration that *IR* size will be checked, then the smallest *IR* will migrate to peer slave to reduce the communication overhead. The peer slave receives a message 'JOIN_IR' from the *Scheduler* to begin the joining whenever it accommodates both the *local IR* and the *peer IR*. Then *Enhanced Sort Merge* takes place. The continuous iteration stops only when the *Scheduler* sends the message 'FINAL_RESULTS' to *slave*. Then the *slave* will send the final *IR* to the *Query Manager*, for better understanding of the flow of interaction between processes, these message tags are shown in *Fig. 10*.

An example of query execution procedure based on the *Parallel Query Algorithm*, which divided the initial query into six sub-queries, is shown in *Fig. 12*. The execution of such procedures is susceptible to delays that arise when retrieving data from workstations because of the different workload on each host and the overload is not constant because those hosts are not dedicated hosts. PQA reacts to such delays by dynamic rescheduling when a delay is detected using *Scheduler* and *Slave* algorithms [11] which exchange messages at run time [13].

For example the initial execution procedure for Q5 is shown in *Fig. 12a*, but *Fig. 12b* shows a different execution procedure for Q5. Relation C_ is not ready to send their intermediate result but relation N_ is finished then PQA received the acknowledgment form that host and at the run time sends a command to the host which holds S_ (as N_ is smaller then that would reduce the communication cost) to receive the IR from N_.



(a) Execution plan for Q5

(b) other Execution plan for Q5

Fig. 12.

4.4. Data Placement Algorithm

Data placement in ESA shows similarities with data fragmentation in *distributed databases.* An obvious similarity is that fragmentation can be used to increase parallelism.

Another similarity is that since the data is much larger than applications, applications should be executed as much as possible where the data resides. However, there are two important differences with the distribution database approach. *First*, there is no need to maximise local processing (at each node) since users are not associated with particular nodes. *Second*, load balancing is much more difficult to achieve in the presence of a large number of nodes. (e.g. one node ends up doing all the work while the other remains idle). Ndiaye YAKHAM et al. in [14] says that parallel DBMS offers at present only static partitioning schemes.

Adding a storage node is then a heavy operation that typically requires the manual redistribution of data. The aim of *Data Placement algorithm*, see *Fig. 13*, is to avoid data skew which deteriorates the system performance by partitioning the relations horizontally into equal sizes, then allocating them to different ESA environments which might be 3, 4, 5, 6, 7 or 8 clustered workstations to achieve maximum performance and minimum utilisation of the resources

4.5. Fault Tolerance in PQA

Robustness in PQA is clarified by having fault-tolerance feature. As explained in [11], the cluster of general-purpose workstations (ESA) in any small organization can be used as separate data server from the original server which the data was obtained from.

There are many types of failures that can occur. For example, a host can fail, a network connection could fail, or a disk could fail. The master host is responsible

```
Assumption:
P    is the number of workstations
Ni   the size of the relation, where i 1 to K
Nt   the total size of the relations
RF   the chunk amount of that fits in the workstation
LCT  the Largest Current Table
RFF  Records to Transfer
CW   Current Workstations

Let LCT = 0 // initialization variable
For (CW = 1 to P) // starting loop
        RTT = RF
// move the right chunk of data in to variable
    IF (LCT = 0) THEN
        LCT = the largest current avalilable relation
// get the largest tables from the DB scheme
    WHILE (CW not full) DO
        IF (Size (LCT >= RTT) THEN
            Allocate RTT records to CW
// place the data into current workstation
            Declare CN as full
            Size (LCT) = Size (LCT) - RTT
// get the remained data from the largest table
            BREAK
        ELSE
            IF (Size (LCT) < RTT) THEN
                Allocate Size (LCT) to CW
// place the data into current workstation
            RTT = RTT - Size(LCT)
// get the remained data to full the workstation
            LCT = the next largest avalilable relation
    ENDIF
    END WHILE
ENDFOR
```

Fig. 13. Data Placement Algorithm

for detecting the failure of a slave and invoking the appropriate recovery actions. Therefore, PVM will detect it, and send a notification message to PQA (which exists in the master host and is controlling all the clustered hosts in Master-Slaves fashion) by using pvm_notify(). When a task is spawned, the TID (task identifier) is kept and passed to pvm_notify(). PVM then sends a massage back to the caller if a failure has been detected. This message has a tag '*msgtag*' to be used in notifications such as '*PvmTaskExit, PvmHostDelete*' which identifies that the task is killed or the host

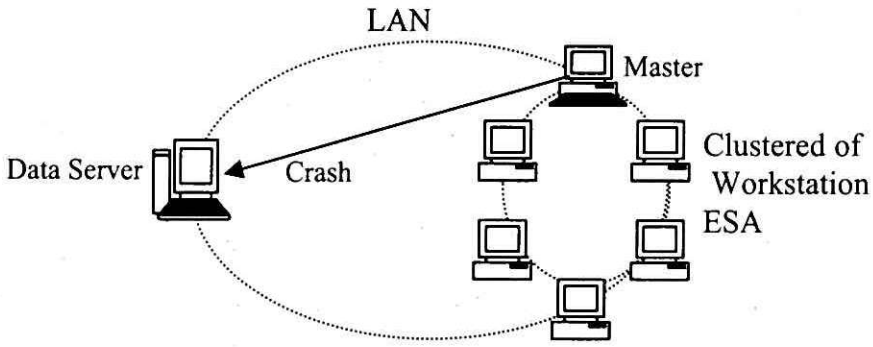


Fig. 14. Fault Tolerance in PQA

has crashed, respectively, as it is explained in [13] in more detail. *Fig. 15* explains the procedure of monitoring ESA, when the failure or crash occurs in ESA. PVM will notify the master where the pvm_notify() exist, then the master will divert the query execution to the original data sever as shown in *Fig. 14.*

```
While i <= NUMHOST \\ Number of hosts
    {... .  .
        cc=pvm_spawn (slaveName, 0, PvmTaskHost,
        hosts[i],1,&tid[i]); if (cc == 1)
                pvm_notify (PvmHostDelete,TASKDELETE,1,
                &tid[i]);
            .  .  .  .  }
            .  .  .  .
While i <= NUMHOST \\ Number of hosts
    {
        buf_id = pvm_recv(-1, -1);
        pvm_bufinfo (buf_id,&msg_len,msg_tag,&msg_src);
        if ( msg_tag == TASKDELETE )
            { ..  ..
                ''here the msg_tag tells the PQA (master)
                that a host is deleted or crashed,
                then divert query execution to the
                original data server.''
                .  .  .  .  .
            }
```

Fig. 15. Implementation of Fault-Tolerance in PQA

5. Experimental Environment and Performance

The graph shown in the following Section tends to exhibit the performance of the PQA approach. The graphs were generated as follows:

First, the TPC-H benchmark databases sets and some of their queries Q3 Q5, were used. *Second,* POSTGRES is used because it is well suited for handling massive amounts of data. Moreover, it also supports large objects that allow attributes to span multiple pages and contains a generalised storage structure that supports huge capacity storage devices as tertiary memory and also it's free, it can be downloaded from [19]. *Third,* The commercial parallel systems are very expensive, thus in this experiment a *Virtual Parallel Machine* (PVM) is used to create a cluster of eight workstations. This provides a cost-effective solution for small businesses. *Fourth,* the experiments were performed in two different environments:

- The *Expandable Server Architecture* (ESA).
- Data Server which is a single workstation with imbedded Postgresql.

In order to effectively measure queries performance in a distributed environment, it is necessary to have a reasonably accurate method that measures the response time.

5.1. Method for Measuring the Response Time

In this experiment, static data distribution is being used in ESA and dynamic scheduling as in PQA. The cost model we used was the *response time* [1]. The response time of a query is defined to be the time elapsed from the initiation of query execution until the time that the last tuple of the query result is completed. If all operators of a plan are executed sequentially, then the response time of a query is added up into the total cost. However, when parallelism is exploited, then the response time of a query can be lower than the one in sequential execution. In this Section, the calculation of response time for entire query is introduced.

Query evaluation in Parallel Database System (PDBS) is quite different from evaluation in sequential systems. Exploitation of parallel systems requires additional tasks and concepts like inter-process communication, scheduling, load balance and parallel implementation of algebra operator [11].

In order to effectively measure queries performance in a distributed environment it is necessary to have a reasonable accurate measuring model. *Response time for query*: the response time of a set of parallel operators is that of the longest one.

When a query is decomposed into sub-queries for example, consider a query that involve five different base relations allocated at five different sites, such as the one in *Fig. 14.* The query is decomposed into sub-queries as described in Section 4.2.2. There are 4 parts for that query, and they consist of PS1, PS2 and PJ1 in part one. In part two there are PS3, PS4 and PJ2. Part three has part two and

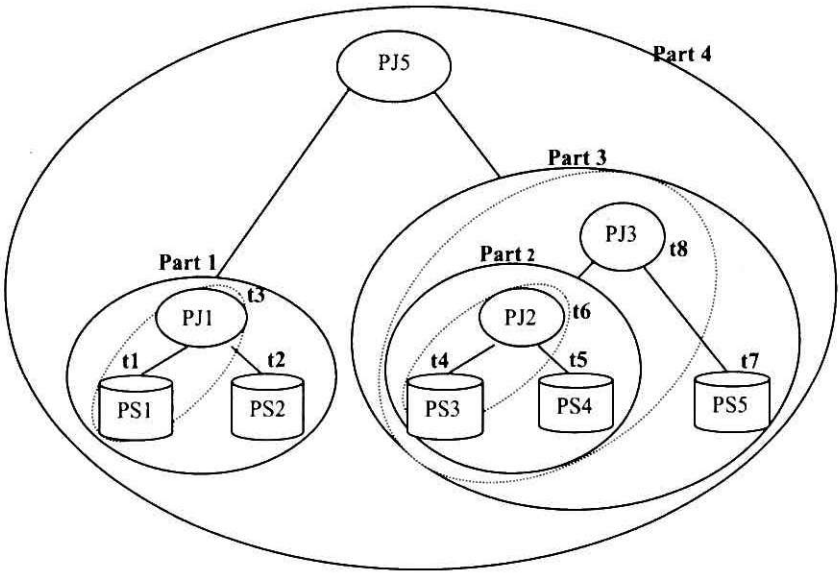


Fig. 16. Measuring the Response Time

PS5, PJ3. Part four consists of part three and part one. PS_j is defined as the time for scanning the disk where $j = 1$ to 5 and PJ_i is the time for joining two intermediate results where $i = 1$ to 4. T_k is the elapsed time to finish the task where $k = 1$ to 9. The response time: In the example shown in *Fig. 16*, there are four parts and the response time *Res_Time* can be calculated by starting with part one and ending with part four including the root operation. Thus we have

$T_{root} = T_{lift}, T_{right})$

$T_{root} = Max\{ IRi \cdot [Log\ 2(IRi)] , IRj \cdot [Log\ 2(IRj)] \} + IRi + IRj$

T_{lift} or T_{right} = Time for Local Processing + Time Communication + Time for Joining Intermediate Relation.

Time IO = (number of tracks per cylinder · Sectors per track · 512) / (2 · Number of surfaces· Latency + (Number of surfaces - 1)· Head Switch Time + Cylinder Switch Time).

Time for Joining IR = $Max\{ IRi \cdot [Log\ 2(Ri)] , IRj \cdot [Log2(IRj)] \}$ + IRi + IRj

5.2. *Performance Evaluation*

The performance of PQA over queries Q3 and Q5 on different ESA environments has been measured. ESA environments for Q5 are 6, 7 and 8 hosts and for Q3 4, 5, 6 and 7 hosts. For the performance comparison Q3 and Q5 are applied on the

original data-server. *Fig. 17* shows the summery of the experiments. The response time is decreased when the number of the hosts is increased because the workload is being tuned.

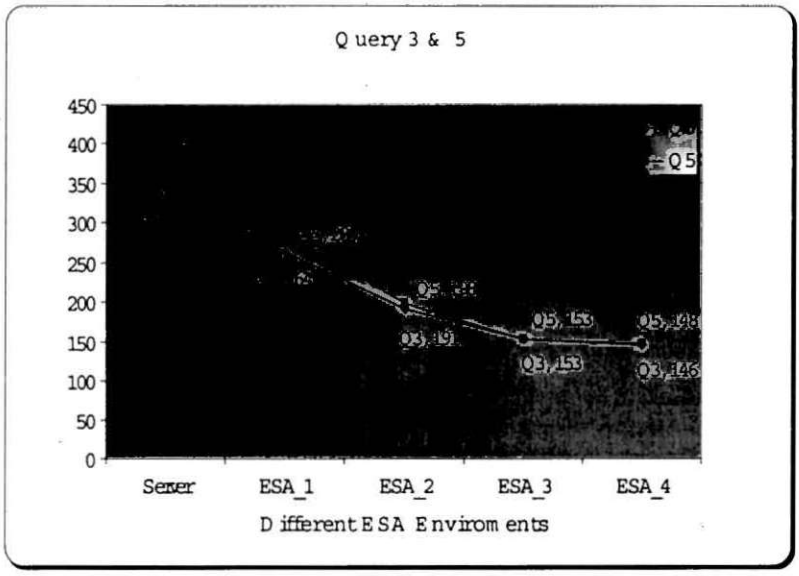


Fig. 17. Query 5 & 3 applied on data-server and different ESA environments

The execution procedure of Query 5 on ESA_1 is outlined below, query 5 is 5-way join query of large and small tables, with selection on table Region, Order, Lineitem and Customer.

```
SELECT   N_NAME, L_EXTENDEDPRICE, L_DISCOUNT
FROM     C_, O_, L_, S_, N_, R_
WHERE    C_CUSTKEY = O_CUSTKEY
         AND O_ORDERKEY = L_ORDERKEY
         AND C_NATIONKEY = S_NATIONKEY
         AND S_NATIONKEY = N_NATIONKEY
         AND N_REGIONKEY = R_REGIONKEY
         AND R_NAME = 'ASIA'
         AND O_ORDERDATE >= '1994-01-01'
         AND O_ORDERDATE < '1994-10-01'
         AND L_SHIPDATE <'1995-03-15'
         AND C_CUSTKEY > 82000
         AND C_CUSTKEY < 84000
```

Query execution environment consists of six different tables C_, O_, L_, S_, N_ and R_ allocated into six different workstations. An example of query execution procedure was based on the *Decompose* algorithm, which divided the initial query

into six sub-queries, showed in *Fig. 18.*

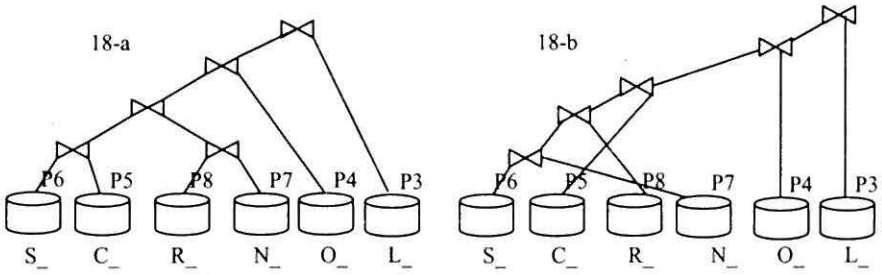


Fig. 18. (a,b) Query execution procedures (Plan) for Q5

The execution of such procedures is susceptible to delays that arise when fetching data from workstations because of the different workload on each workstation. PQA reacts to such delays by dynamic reschedule when a delay is detected using *Query Manager* and *Slave* algorithms which exchange messages at run time. For example the initial execution procedure for Q5 is shown in *Fig. 17a*, but *Fig. 17b* shows a different execution plan for Q5 by the time when Q5 has executed it, due to delay that occurs and the dynamic scheduling takes place.

The workstations memory is used to temporarily store the intermediate subqueries resulting in structure of arrays and then ship them to the corresponding workstation. For example, the intermediate result of workstation 6 (P6), which holds the relation S_, is 10000 tuples with size of 4 bytes each, about 40000 bytes. And workstation 5 (P5) which holds the relation C_, is 1999 tuples with size of 8 bytes each, about 15992 bytes. Workstation 8 (P8) which holds the relation R_, is only have one tuple with size of 29 bytes. As for workstation 7 (P7), which holds the relation, N_, are 25 tuples with size of 8 bytes, about 200 bytes. Workstation 4 (P4), which holds the relation O_, is 170378 tuples with size of 8 bytes each, about 1363024 bytes. As for the largest relation L_ which exists in workstation 3 (P3), are 2756911 tuples with size of 22 bytes, about 66165864 bytes. Due to different workload in the workstations and some other reasons, which are discussed in Section 4.3, PQA dynamically reschedules the plan.

Paging in the Receive buffers memory space in the workstations causes the large increase in data transfer time above 16 Mbytes per workstation. The next physical limitation as the intermediate result size increases beyond 16 Mbytes is the size of the swap file used for page-swapping, since our system operates in the virtual memory of the workstations. The size of the swap file can be increased up to the limitation of the available disk space accessible to each workstation. The swap file can be placed on any mounted drive, but a computer can slow down dramatically if a workstation is used for virtual memory swap space. Therefore, data-server performance is slower than any ESA environment due to time spent by the data-server optimiser trying to find the best execution plan and to sequential data retrieval.

6. Conclusion

The progressively increasing computing power and memory space of successive generations of general-purpose workstations is creating a potential hardware resource for parallel processing. The *parallel virtual machine* (PVM) system is a software package which enables message passing between computers and so helps to create a 'Parallel Virtual Machine' out of these hardware resources. (PVM) is easy to install and use and supports heterogeneity both at the machine and network levels. PVM is a dynamic configuration; it can add and delete processes at execution time and at any point in the execution of concurrent applications, the processes may communicate with and synchronise each other. But a main disadvantage of (PVM) is the lack of an accurate debug facility.

The scalability of message passing in (PVM) was investigated by sending database tables of progressively increasing size to a virtual machine system (a cluster of eight workstations) and then to a single workstation. The results of the investigation showed that the virtual machine system was faster than the local system but the speed varied with database size.

Performance also declines with increasing table size, till certain data size is reached. The study revealed that over a particular data size the performance of transferring database tables decreased due to the page-swapping mechanism taking place.

Workstations in the same local network as the data server can provide a dynamically extensible and reconfigurable computing resource for rule derivation and maintenance for *Semantic Query Optimisation*. This additional resource is provided by utilizing existing hardware, workstations which are used for computationally undemanding tasks which form the usual workload of desktop computers. The work involved in rule derivation and maintenance is thus removed from the data server onto other workstations, described in Section 3. The master workstation can measure the workload on each workstation in the network by spawning a short program on the workstation, and measuring the time it takes to finish. Its runtime under various computer workloads is known. So the measured time indicates the workstation's current workload and thus its suitability as a place to run a new rule derivation task.

Deriving histogram rule set is a way to detect subset dependencies in data. The ability to rapidly derive rule sets from data therefore makes this aspect of data analysis easier. It allows the potential usefulness of rules to be quickly recognized and prevents fruitless attempts to produce rules from data which does not support them. The scanning algorithm discussed in Section 4 is amenable to parallel implementation by either horizontal or vertical partitioning of database tables. The effect, in either case, is the simultaneous derivation of N histogram rule sets by partitioning to N workstations. Vertical partitioning, assigning different pairs of columns to different workstations, gives slightly slower rule set derivation. But it also has the more significant drawback that if the data is subsequently sorted, the operation will be very slow. It requires the one-workstation time rather than the

N-workstation time described in Section 3.2.

Experimental results for sorting data on multiple workstations show a useful sublinear speedup. The effect of sorting by antecedent attribute value is to cluster tuples for each rule antecedent; therefore sorted data allows direct access to the data subset selected by a rule's antecedent condition. Descriptors for that subset can be revised, following data changes. A choice must be made about whether to derive rules by the sorting or the scanning algorithm. For 'small' tables, the sorting algorithm can be completed rapidly, if enough workstations are used, so sorted data as well as a rule set is immediately available. However, the scanning algorithm is faster than the sorting algorithm and the difference becomes increasingly significant as the amount of data per workstation increases. The experimental results suggest that the scanning algorithm should do the initial derivation of each set of subset descriptor rules, unless the table is small (less than 150 000 rows) and at least 9 slave workstations are available. This makes rules available for query optimisation as quickly as possible at the time they are needed.

Data in the slaves can be sorted *after* rule derivation, to support rule maintenance.

In an ordinary local network bandwidth is limited and data transfer is necessarily sequential. Distributing subsets to workstations therefore takes an amount of time related to the size of the database table. It is not possible to send different subsets simultaneously from the master workstation. Therefore the time to create a rule set must increase to some extent as table size increases, because of the time needed to copy the table into the workstations. The sorted raw data in multiple workstations can also provide rapid *data retrieval* for database queries or sub-queries, and this facility can be utilized by the 'master' workstation query interface when deciding the quickest way to answer each query. Some queries will be re-written by semantic query optimisation methods using the information provided in the subset descriptor rules. These re-written queries will then be sent to the DBMS server to answer. Other queries will be decomposed into sub-queries for distributed query processing on some combination of workstations and DBMS data server. Therefore, *parallel query algorithm* (PQA) was designed and developed along with *expandable server architecture* (ESA), described in Section 4.

The performance of the parallel query processing algorithm (PQA) was examined on a single computer and (the ESA_x), where x is different environment of ESA results) and found to give better query processing speed than executing the same query without the parallel algorithm.

The architecture of the Expandable Server LAN system is conceptually and behaviorally between that of a multi-processor database server and a wide area network Distributed Database. All three architectures have multiple processors, but the character of the interconnection network affects the way they can be used. Data transfer time on an Ethernet LAN can become significant if large data sets are being transferred. This affects the ways that the expandable server architecture can be used. Tasks can be allocated to computers, which contain the data tables relevant to that task. Computers must contain the data *before* tasks are allocated to them, unless the data set is small, because the time required to transfer data between machines

can outweigh the time benefits provided by parallel processing. Therefore, Data Placement Algorithm tune the workload the initial strategy for allocating data sets to computers will affect the scope for workload in ESA environments, described in Section 4.4. Replication of data sets is desirable, and pairs of subsets should be allocated to the same computer if they are to be joined (so that network traffic is reduced). The execution of a query plan is susceptible to delays that arise when retrieving data from workstations because of the different workload on each host, and the overload is not constant because those hosts are not dedicated hosts. PQA reacts to such delays by *dynamic rescheduling* when a delay is detected using *Scheduler* and *Slave* algorithms which exchange messages at run time, described in Section 4.3. When failure or crash occurs in ESA, PVM will notify the *master* where the pvm_notify() exists, then the *master* will divert the query execution to the original data server. There is clearly a limit to the size of database that can be distributed to desktop computers, because of the limited storage space usually available on general-purpose computers. The number of gigabytes per machine is steadily increasing, but Data Warehouse table sizes, for example, are by orders of magnitude larger, so that even when partitioned to a number of machines the table fragments are too large.

References

[1] CHEN, W. K., *Applied Graph Theory*. New York: American Elsevier Publishing Co., and Amsterdam, The Netherlands: North-Holland Publishing Co., 484 pp., 207 illustrations, 1971.

[2] DEWITT, D. J. – GRAY, J., Parallel Database Systems: The Future of High Perfomance Database Systems, *Communications of the ACM*, **35** (6) (June, 1992).

[3] GEIST, A. – BEGUELIN, A. – DONGARRA, J. – JIANG, W. – MANCHEK, R. – SUNDERAM, V., , *PVM.: Parallel Virtual Machine A users' Guide and Tutorial for Networked Parallel Computing*, ed. J Kowalik, MIT Press (1994) Also available on-line: 'http://www.netlib.org/pvm3/book/pvm-book.html' or via anonymous FTP from ftp.netlib.org.

[4] LU, H. – OOI, B.-C. – TAN, K. L., *Query Processing in Parallel Database Systems*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.

[5] ROBINSON, J. – LOWDEN, B. G. T. – ALHADADD, M., *Distributing the Derivation and Maintenance of Subset Descriptor Rules*, The 5th World Multi-Conference on Systemics, Cybernetics and Informatics. SCI 2001. July 22–25, 2001. Orlando, Florida USA.

[6] LAKSHMI, M. S. – YU, P. S., Effectiveness of Parallel Joins, *IEEE Transactions on Knowledge and Data Engineering*, **2** (4) (December 1990).

[7] LIU K. H. – JIANG, Y. – LEUNG, C. H. C., Query Execution in the Presence of Data Skew in Parallel Databases, *Australian Computer Science Communications*, **18** (2) (1996), pp. 157–166.

[8] MUTKA, M. – LIVNY, M., The Available Capacity of a Privately Owned Workstation Environment, *Performance Evaluation*, **12** (4) (July 1991), pp. 269–284.

[9] OLSON, M. A. – HONG, W. M. – STONERBRAKER, M., Query Processing in a Parallel Object-Relational Database System, *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, IEEE 12/1996.

[10] POESS, M. – FLOYD, C., New TPC Benchmarks for Decision Support and Web Commerce, *ACM SIGMOD Record*, **29** (4) (December 2000).

[11] ALHADDAD, M. – ROBINSON, J., Using a Network of Workstations to Enhance Database Query Processing Performance, *Proc. 8th European PVM/MPI 2001 Conference*, Page 352–359, Lecture Notes in Computer Science LNCS 2131.

[12] ALHADADD, M. – ROBINSON, J. – COLLEY, M., Extending Database Technology by Expanding Data Server, *the 6th World Multi-Conference on Systemics, Cyberntics and Informatics*, SCI 2002, July 14–18 2002, Orlando, Florida USA.

[13] ALHADDAD, M. – COLLEY, M., Parallel Query Algorithm Performance and Fault Tolerance, DATAKON 2002 Database Conference, October 19–22, Czech Republic.

[14] YAKHAM, N. – DIENE, W. – LITWIN, A. – RISCH, W., AMOS-SDDS: A Scalable Distributed Data Manager for Windows Multicomputers To be presented at the *ISCA 14th Intl. Conf. on Par. and Distr. Computing Systems*, Texas, USA, August 8–10, 2001.

[15] CHENG, Q. – GRYZ, J. – KOO, F. – LEUNG, C. – LIU, L. – QIAN, X. – SCHIEFER, B., Implementation of Two Semantic Query Optimization Techniques in DB2 Universal Database. *Proc. of VLDB*, pp. 687–698.

[16] ROBINSON, J. – LOWDEN, B. – MOHAMMED, A., Utilizing Multiple Computers in Database Query Processing and Descriptor Rule Management, Dexa'01 September 3–7 2001, LNCS 2113, page 897.

[17] SHEKAR, S. – STRIVASTAVA, J. – DUTTA, S., A Formal Model of Trade-off Between Optimization and Execution Costs in Semantic Query Optimization, *Proc. 14th VLDB*, Los Angeles, CA, pp. 457–467.

[18] DANDAMUDI, S. P. – JAIN, G., Architectures for Parallel Query Processing on Networks of Workstations, *Proc. Int. Conf. Parallel and Distributed Computing Systems*, New Orleans, (October 1997).

[19] LOCKHART, T. The PostgreSQL Administrator's Guide, The Administrator's Guide , 2001-04-13, The PostgreSQL Global Development Group, http://postgresql.readysetnet.com/devel-corner/docs/postgres.

[20] ANDERSON, T. T. – CULLER, D. – PATTERSON, D., A Case for Networks of Workstations: NOW. *IEEE Micro*, Feb. 1995.

[21] ALHADDAD, M., Utilising Networked Workstations to Accelerate Database Queries, *The Third Conference of PhD Students in Computer Science CSCS 2002*, Szeged, Hungary, July 1–4, 2002.