International Journal of Multimedia and Ubiquitous Engineering
Vol. 4, No. 2, April, 2009

Network recourses (NOW) to remove the work load for Deriving rule for Semantic Query Optimization and Speed up answering queries

Mohammed Jaffer Alhaddad
Department of Information Technology
malhaddad@Kau.edu.sal

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 invistagete 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 parallelism in two separate ways: (i) Remove 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 [1,2]. (ii) Answer 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 [1,2,3,4]. Experiments have been conducted to tackle the problems of Parallel and Distributed systems such as task scheduling, load balance and fault tolerance.

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 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 [5]. 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 optimization methods. Such methods as described in this report are by the utilization of a network of workstations.

The goal of this research is to utilize any available computers in a data server's local network to optimize database query processing. Parallel Virtual Machine (PVM) is the software that allows utilization 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. The project goal was pursued in two separate ways as in figure 1.

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

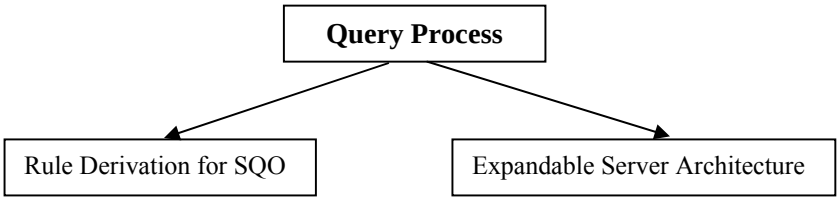

Figure 1. The goal of the research.

The significance of the "Rule Derivation for SQO" component of my research is that it greatly improves the applicability of semantic query optimization 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 optimization, 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.

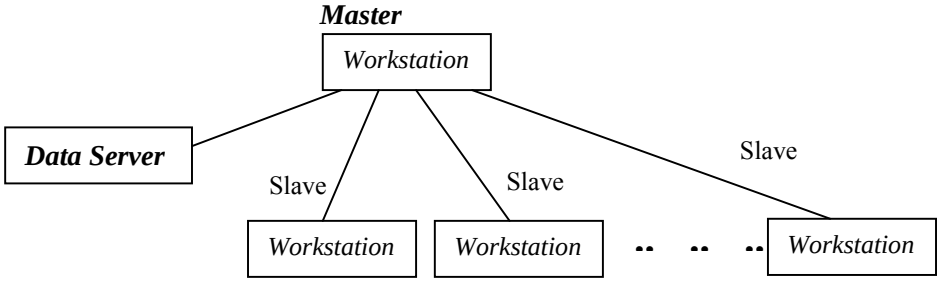

Figure 2. Utilizing Workstations for Semantic Query Optimization.

The demonstration of the performance of the systematic rule-set derivation algorithm, which utilizes multiple workstations, removes this problem, see figure 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 [6]. The author does 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 has 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 case where the sub-query has 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 coordinating 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 [3] implemented on client-server architecture with 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, depend on its participating to answer the original query. At any moment, the computing power of the virtual pool can be fully utilized 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 computers in the local network by using Parallel Virtual Machine PVM [9]. 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 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 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 follow. In section 2 an investigation of using PVM to create a cluster of workstation is conducted. Section 3, explains utilizing cluster of networked of workstation to create a rule set for Semantic Query Optimization. An overview of ESA, PQA, Dynamic schedule allocation and Data placement are given in section 4. Section 5 illustrates the performance and valuation for PQA on ESA; on general purpose workstation and on dedicated workstations, followed by conclusion in section 6

2. Environment and tools of the project

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 organizations 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 a physical memory limitation 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 distribute 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 150000 rows for this 112-bytes-per-row table is the maximum size per host (for these particular hosts) to avoid the delay. 150000*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 graphs.

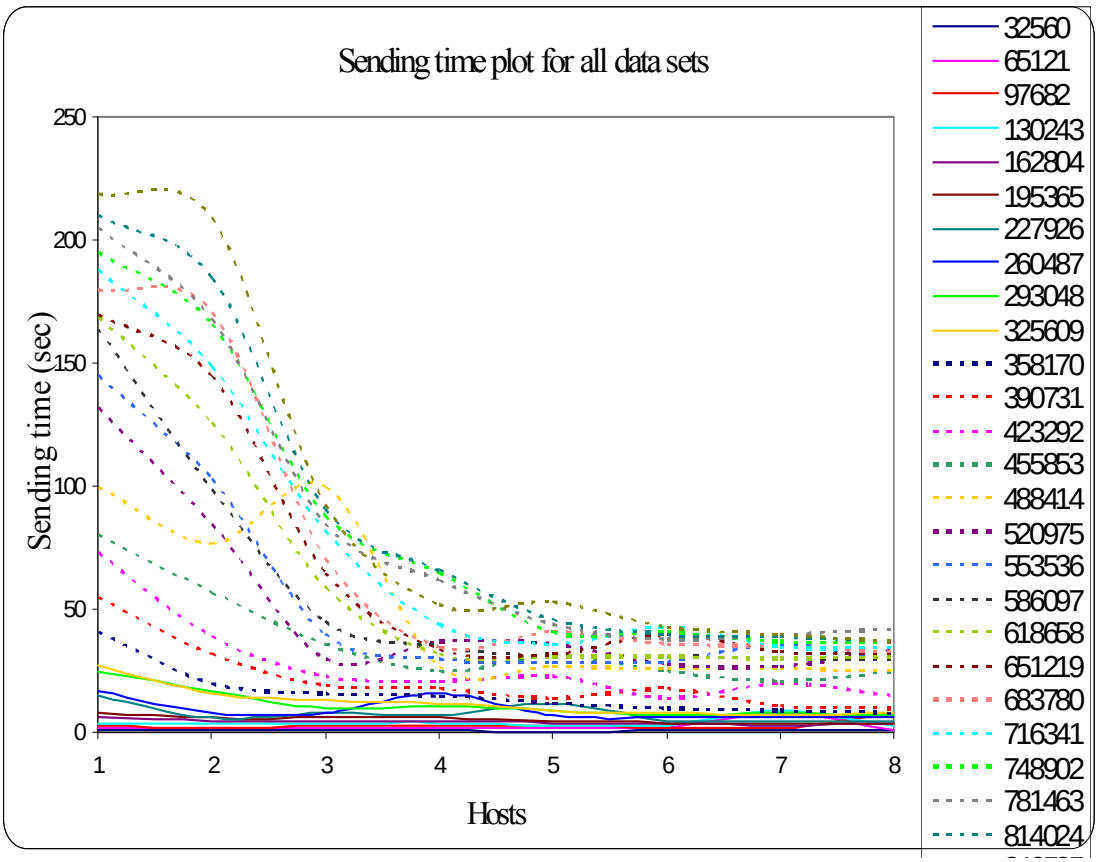

Figure 3. Sending different size of data set over a cluster of 8 hosts

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 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 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.

3. Obstacle in Semantic Query Optimization approach

Semantic query optimization, 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 developed techniques for Semantic Query Optimization. To the best of the author's knowledge, there are no broad commercial implementations of SQO. There are number of reasons for this lack of implementation. First and foremost, because SQO is associated for many years with cases designed for deductive database, it was not thought be useful for other uses such as relational database technology [10]. Second, it has been commonly assumed that for a SQO to be of benefit, many integrity constraints have to be defined for a given database. Otherwise queries could not be optimized semantically. Finally, the speed of the CPU and the I/O at the time when the Semantic Query Optimization developed was different than nowadays. [11] Shows that the cost of semantic optimization could be comparable to the query execution cost.

Semantic Query Optimization 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 author used the attribute pair rule [4]. These rules are also created automatically using QuickSort and Scan Bucket Algorithm for semantic query optimization.

3.1. Sorting Data Subsets for Rulesset Derivation and Maintenance

The data in a Database table or view is partitioned by the Master workstation to 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 Master workstation also tells the Slaves which attribute to use as antecedent for the current ruleset. 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 rulesets derived from a 400000-row table, for example, for up to ten slaves.

When the antecedent attribute is numeric the master tells the slaves how many rules to derive. The master also broadcasts the MIN and MAX values for the attribute so that all slaves 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 also makes rule maintenance easy.

As shown in table 1, the measured time means the observed time taken to create a set of rules (a histogram ruleset) from the 130239-row database table. 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 experimental results show even better speedup, as indicated in Figure 4. (Two machines are more than twice as fast as one, etc). The explanation for this better than predicted performance is partly the computational complexity of the Quicksort algorithm used. It has a best case complexity of NlogN for N data items, and a worst case of N2. 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 data sets do proportionately more page swapping.

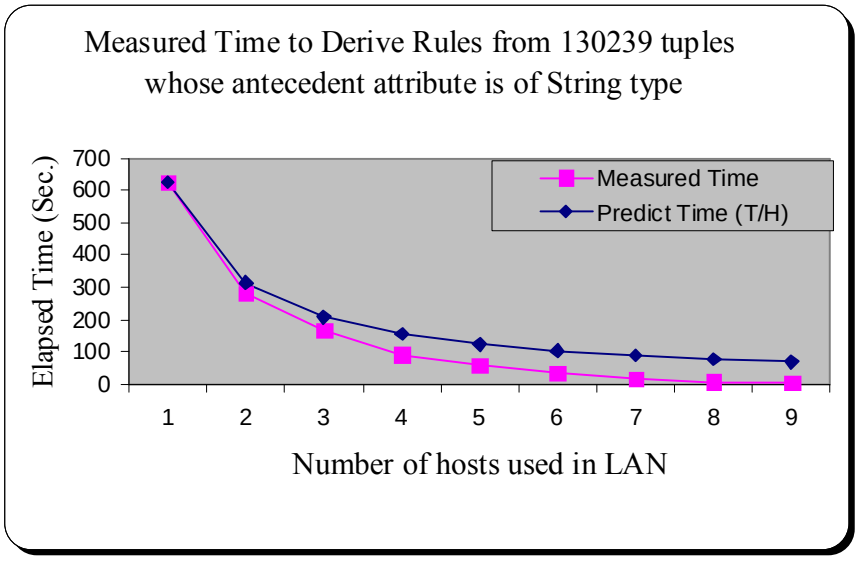

Figure 4. Measure time for rule set derivation shows better than linear speedup

Table 1. Expected time and real experiment measured time

No. of Hosts, H	1	2	3	4	5	6	7	8	9
Measured time (sec)	625	282	167	91	61	36	16	8	5
Expected 625/H (sec)	625	313	208	156	125	104	89	79	70

Figure 4 shows that when 9 workstations are used it took only 5 seconds to distribute and sort the data sub-sets, derive 9 separate histogram rulesets and merge them into a single ruleset 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 optimize the next query. This query-triggering of ruleset 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 is 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 rulesets can be derived in a few seconds by broadcasting only the derivation parameters. These are 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 ruleset 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 optimization 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.

3.2. 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:

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

This indicates that all tuples encountered so far whose attribute 'a' value was in the range $15 \leq \square a \leq \square 30$ 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 ($15 \leq \square a \leq \square 30$). 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.

3.3. 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, is not produced. These measurements, as shown in figure 5, 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.

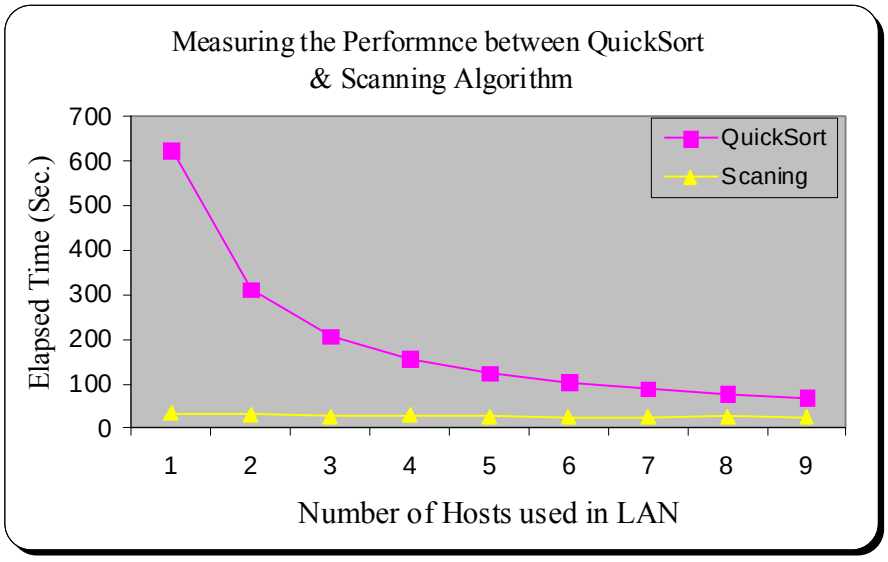

Figure 5 Comparing ruleset Derivation times for QuickSort & Scan Algorithm

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.

3.4. 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 detail and due to space restriction please [5, 9].

4. 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 in 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.

The purpose of querying the database is not only satisfying the query but to minimize 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

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 utilizing 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 [12] there is no central control therefore it is impossible to distribute the database among workstation, the database relations are stored in central workstation. Like Parallel Database Systems, ESA with Parallel Query Algorithm PQA [9] 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

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, as it shown in figure 6. In NOW [14] one workstation has control over the database and the others have access to 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 in not a simulation study as in NOW, a real data about 2GB is being use from TCP-H and Data Placement Algorithm is used to tune static data distribution among the cluster of workstations to achieve an optimal performance.

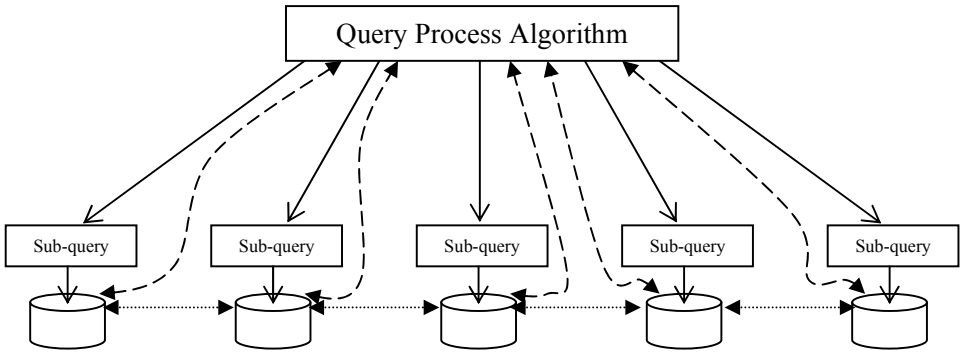

Figure 6. The parallelism in ESA by PQA

4.3. Parallel Query Algorithm Components.

In this section a description of PQA and all it components will be presented. Figure 7 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 initialization 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 in to sub-queries. 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 process 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. 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 is made by the Scheduler and sent to Slave telling it either to send IR in to peer Slave or to receive IR from peer Slave or sort its IR according to join predicate. Since the behavior of the components varies and they receive different acknowledgement messages, a table of message tags is maintained, see table 2. In addition, for better understanding of the flow of interactions between processes, these message tags are shown in figure 7.

Table 2. Message tags

Process	Tag Identifier	Tag	Description
Scheduler	sub-query	10	Send sub-query to salve 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 salve 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.

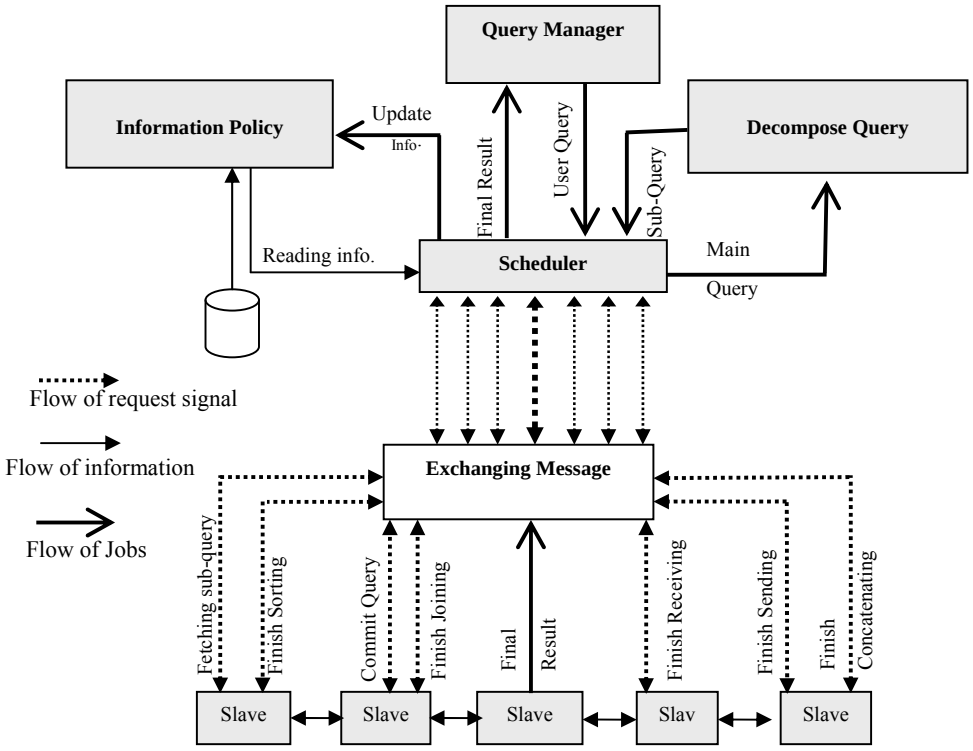

Figure 7. Flow of Messages in PQA

4.4. Dynamic Load Scheduling.

A query evaluation plan is generated by the optimizer. The task of the optimization 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 optimization 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 optimization steps into the execution phase can alleviate the problem of optimization in parallel database systems.

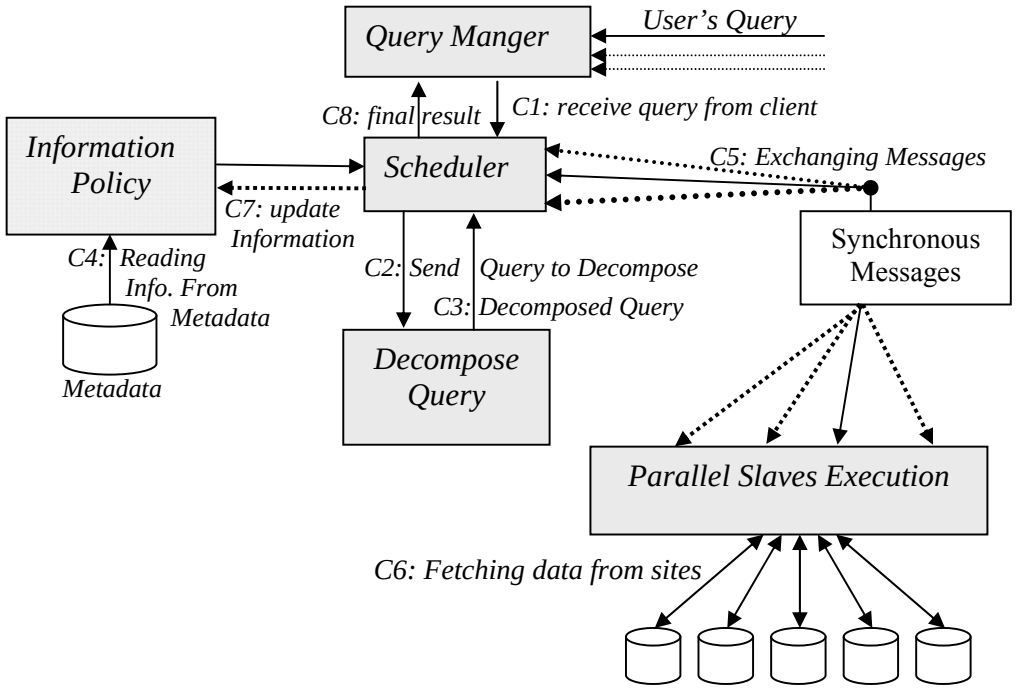

Figure 8. Parallel Query Algorithms PQA.

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 figure 8, an arbitrary processing node "Query Manager" receives it and becomes the co-coordinator in charge of optimizing 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 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 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 figure 7. The dynamic scheduling of tasks at run time is begun when a slave receives the message "FETCH SUBQUERY" to fetch a sub-query, then the Intermediate Relation is obtained and 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 over head. 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 figure 7.

An example of query execution procedure based on the Parallel Query Algorithm, which divided the initial query into six sub queries, shows in figure above. The execution of such procedure 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 delay by dynamic rescheduling when a delay is detected using Scheduler and Slave algorithms [3] which they exchange messages at run time [15].

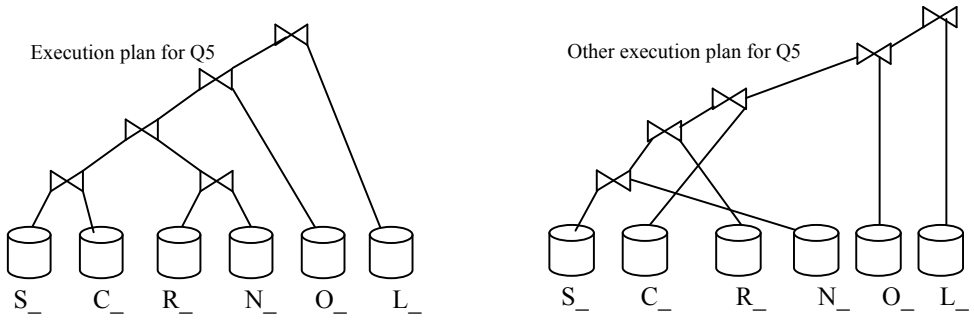

Figure 9. Dynamic Load Scheduling in PQA

For example the initial execution procedure for Q5 is shown in figure 9; however figure 9 shows other 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_.

4.5. Data Placement.

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

```
   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 available relation  //
   get the largest tables form 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 available relation
     ENDIF
     END WHILE
   ENDFOR
```

Figure 10. Data Placement Algorithm

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 maximize 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 [13] said 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, as shown in figure 10, 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 utilization of the resources

5. Performance Evaluation for PQA on ESA: –on general purpose workstation and on dedicated workstations

The response time of a query is defined to be the elapsed time from the initiation of query execution until the time that the last tuple of the query result is completed.

An experiment was carried out at an open resources lab accessible by wide variety of users such as students, staff, researchers and system administrators (back up and maintenance routines). Therefore, the respond time for Q5 executed in different time in a day is variable as shown in figure 11.

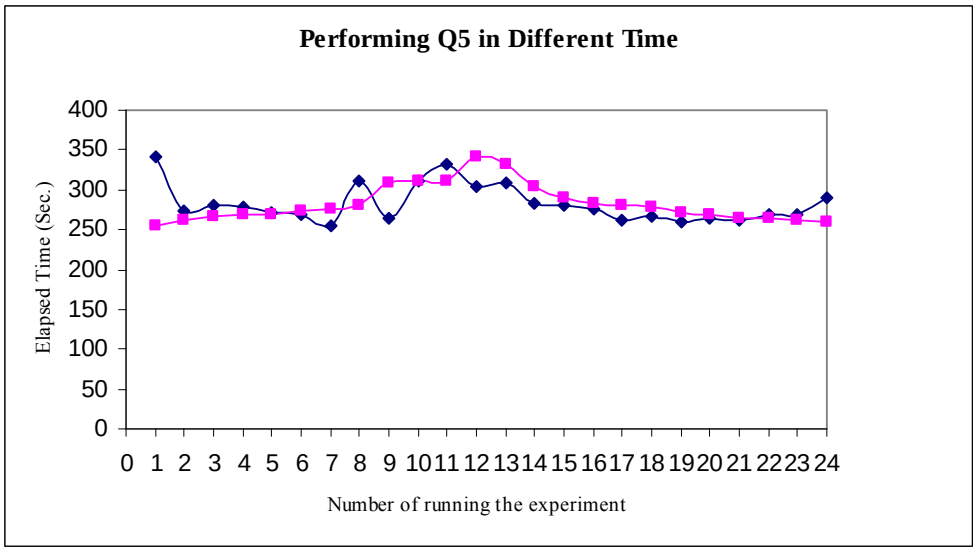

Figure 11. Running query 5 in different time of the day

Figure 12 shown 4 different types of environments, as follows: First - the response time of executing Query 5 on the Expandable Server Architecture ESA shows that the final response time decreased in comparison with that of the PSQL, which is the case of one workstation without the support of PQA. As in the ESA environment, the sub-queries were executed in parallel which saved considerable time in comparison with the other environments. Second - by using PQA over a central database on a single workstation Single_PQA produced a better performance than PSQL. In the case where one workstation was used with the support of PQA, the sub-queries were executed sequentially and the time of each execution was added up to give the overall response time to the main query and there is no time wasted for the query optimizer to generate the optimum query. While in the case of PSQL, the time was

spent on the query process trying to find the best strategy plan for the execution of the query and on sequential access to the data thus taking the most time. Third – the performance of PQA over a cluster of dedicated workstation D_ESA showed the best response time because there is no back ground process burden the performance.

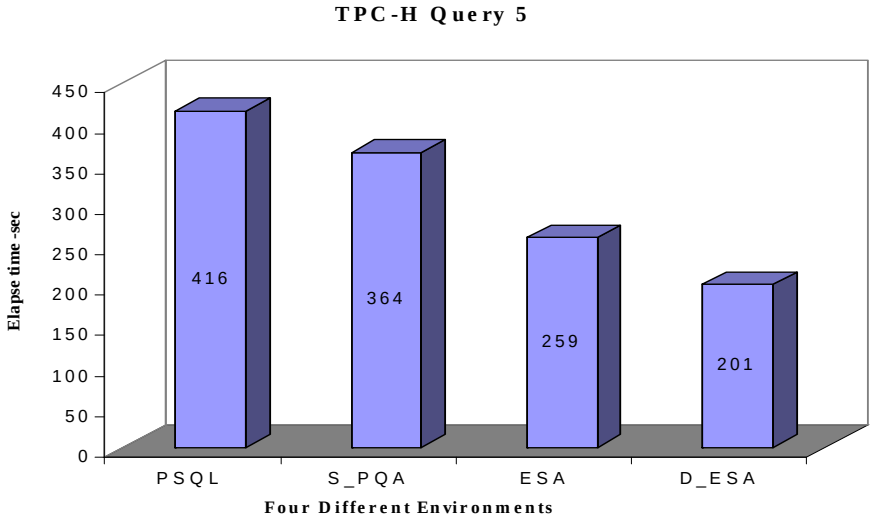

Figure 12. Performance of PQA in four Different Environments

6. Conclusion

In this project, PVM has clearly demonstrated its ability to use networked workstations as a single computational resource able to exploit parallelism. Experiments of measuring the scalability of PVM show that there is a limitation to the size of data that can be sent to one workstation. This limitation is caused by the restricted size of the memory that can hold the data at one time, which increases the sending time as page swapping will take place. The sending time is thus reduced as the number of workstations is increased and the size of data is decreased, this allows the data to be distributed and dealt with in a shorter time

The significance of the Rule Derivation for Semantic Query Optimization (SQO) is that it greatly improves the applicability of semantic query optimization 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 costs. However, the task of examining tables of data to derive sets of rules can be a significant workload. The scanning algorithm is amenable to parallel implementation by either horizontal or vertical partitioning of database tables. The effect, in both cases, 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 a slightly slower ruleset derivation. However, it also has the more significant drawback that if the data is subsequently sorted the operation will be very slow. Experimental results for sorting data (by QuickSort Algorithm) on multiple workstations show a useful sub-linear 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 "QuickSort Algorithm" or the Bucket Scanning algorithm. For 'small' tables the sorting algorithm can be completed rapidly if enough workstations are used, so sorted data, as well as a ruleset, is immediately available.

However, the Bucket 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 150000 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. 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 ruleset describing the changed database table in less than 2 seconds by this method.

As for answering user's queries, in general, in such networked of workstations environment's the I/O resources could have a great effect on the performance. Firstly, the internal factors which include: mechanical design (size and number of platter surfaces, actuator and spindle motor speed) and data recording (track and sector layout). Secondly, the external factors such as disk interface (interface type, CPU utilization and command overhead. Finally, the operating system, caching and virtual memory have an effect over all the performance. Measuring the performance during a day of Q5 using PQA has shown a variable respond time using PQA over ESA. Whereas D_ESA (Devoted networked of workstation) has taken the network congestion and back ground process noise away and obtaining better respond time.

References

[1] Mohammed Alhaddad, "Analysis Parallel Query Algorithm performance and efficiency by devoted networked workstation", International Symposium on Computer Science and its Applications, CSA-08, proceedings published by IEEE CS, Hobart, Australia, October 13 - 15, 2008.

[2] Robinson J, Lowden B, Alhaddad Mohammed, "*Distributing the Derivation and Maintenance of Subset Descriptor Rules* ",The 5 th World Multi-Conference on Systemics, Cybernetics and Informatics. SCI 2001. July 22-25, 2001. Orlando, Florida USA.

[3] Mohammed Al Haddad, Jerome Robinson, "*Using A Network of workstations to enhance Database Query Processing Performance*", Euro PVM/MPI 2001, The 5th World Multi-Conference on Systemics, Cybernetics and Informatics, July 22,2001, LNCS 2131 Page 352.

[4] Robinson J, Lowden B, Alhaddad Mohammed. "*Utilizing Multiple Computers in Database Query Processing and Descriptor Rule Management*". Dexa'01 September 3-7 2001, LNCS 2113, page 897.

[5] M. Mutka and M. Livny.: The Available Capacity of a Privately Owned Workstation Environment. Performance Evaluation, Vol. 12, No. 4 (July 1991) pp. 269-284

[6] Meikel Poess, Chris Floyd: "New TPC Benchmarks for Decision Support and Web Commerce", ACM SIGMOD Record, 29(4) December 2000.

[7] Liu K. H., Y. Jiang and C. H. C. Leung, "Query execution in the presence of data skew in parallel databases", Australian Computer Science Communications, Vol. 18, No. 2, 1996, pp157-166.

[8] Mohammed Alhaddad Robinson J, "Extending Database Technology by Expanding Data Servers ",The 6 th World Multi-Conference on Systemics, Cybernetics and Informatics. SCI 2002. July 14-18, 2002. Orlando, Florida USA.

[9] Mohmmed Alhadadd, Jerome Robinson, Martin Colley: "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.
[10] Q. Cheng, J. Gryz, F. Koo, C. Leung, L. Liu, X. Qian, and B. Schiefer, (1999): Implementation of two semantic query optimization techniques in DB2 universal database. Proc. of VLDB, Pages 687—698.
[11] S.Shekar, J.Strivastava, and S. Dutta. (1988): A formal model of trade-off between optimization and execution costs in semantic query optimization, Proc. 14th VLDB, Page 457-467, Los Angeles, CA,.
[12] Sivarama P. Dandamudi and Gautam Jain.: Architectures for Parallel Query Processing on Networks of Workstations". Proc. Int. Conf. Parallel and Distributed Computing Systems, New Orleans, (October 1997).
[13] Ndiaye Yakham,Wane Diene, A., Litwin, W., Risch, 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.
[14] Thomas T. Anderson, D. Culler, and D. Patterson. A Case for Networks of Workstations: NOW". IEEE Micro, Feb. 1995.
[15] Mohammed Alhaddad, Martin Colley "*Parallel Query Algorithm performance and Fault Tolerance*" , DATAKON 2002 Database conference, October 19-22, Czech Republic.

Authors

Mohammed J. Alhaddad has received his Bsc in computer science in 1986 and his master in 1999 and his PhD in 2006. He was working in MIS and NCB.
He is acting vice dean of admission & Registration in the North Border university in KSA, chairman of Information Technology department in King Abdul Aziz university and chairman of the Computer Science & IT club at King Abdul-Aziz University.