Using A Network of workstations to enhance Database Query Processing Performance

Mohammed Al Haddad, Jerome Robinson

Department of Computer Science, University of Essex,
Wivenhoe Park, Colchester,
CO4 3SQ, United Kingdom
mjalha@essex.co.uk

Abstract. Query processing in database systems may be improved by applying parallel processing techniques. One reason for improving query response time is to support the increased number queries when databases are made accessible from the Internet.

1 Introduction

Applying parallel processing techniques, like Parallel Query Processing in database systems, will 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.

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. Such method as described in this paper is by the utilization of a network of workstations. It has been observed that, up to 80% of workstations are idle depending on the time of the day [1].

Parallel Virtual Machine is software that allows utilization of networked workstations as a single computational resource.

We present in this paper the effective use of Parallel Virtual Machine in enhancing the performance of Parallel Query Processing, with the use of a proposed Parallel Query Interface PQI, which is explained in section 4.2. We also demonstrate the cost effectiveness of the proposed Expandable Server Architecture ESA, which uses the shared nothing architecture. The shared nothing architecture is a relatively straightforward to implement, and more importantly has demonstrated both scalability and reliability. It has also proved to be a cheap way of connecting processors to build parallel database system that could be affordable to small businesses in contrast to the high cost of the most commercially available parallel database systems.

2 Parallel Architecture

Parallel database systems constitute a combination of database management and parallel systems to increase performance and availability. Parallel systems provide much better price / performance than their mainframe counterparts. These advantages could be categorised as follows: ***high performance*** - obtained by operating system support, parallelism and load balancing-, ***high availability*** - obtained by data replication - and ***Extensibility –*** obtained by smooth expansion of processing and storage power to the system [5].

2.1 Parallel Database System

A parallel database system can be built by connecting together several processors, memory units, and disk devices through a network [2]. Depending upon how the different components are interconnected they can be categorised under three main classes [3]: shared-nothing, shared-disk, and shared-everything. Each architecture consists of processors, memory units, disk units, local buses, and a global interconnections network. In the experiment described in this paper, the shared-nothing architecture is used, where all processors are connected via Ethernet interconnection network.

Because none of the components in the shared-nothing architecture are shared, the need for a complex interconnection network is removed. In contrast, the shared-memory and shared disk systems needs a powerful interconnection network because it has to transmit large quantities of data. The shared-everything and shared-disk architectures are not suitable for large systems due to the interconnection bandwidth limitations. The shared-nothing architecture is popular for its scalability and reliability.

3 Parallel DBMS Techniques

A successful implementation of parallel architecture heaviliy relies on the parallel techniques used. This research mainly focuses on two main components, namely ***partitioning*** and ***query parallelism techniques***.

3.1 Partitioning

In relational database schema, relations (Tables of data items) can be fragmented into a number of parts stored on different disks associated with different processors.
The three most common ways of partitioning the data are: ***Hash partitioning:*** This method is suited for sequential and associative access as the tuples will be placed amongst different fragments using some hash functions mapped on an attribute or attributes of the relation. The advantage of this strategy is that the scan will be directed to only one workstation's disk instead of all of them. ***Range Partitioning***

(clustering): Tuples with the same value of attribute and are frequently accessed are placed together. This method suits sequential and associated access, but clustering can lead to a few problems during that access, such as data skew. ***Round Robin:*** Tuples are distributed in a round robin fashion. The round robin strategy works well when majority of the queries involve the whole relation to be accessed sequentially. But, bottleneck may be created when tuples have a common value for a particular attribute. Relational data base schema has a number of tables, in this experiment the partitioning was created by allocating each table into a different node, as can be seen in section 4.1.

3.2 Query Parallelism

Query parallelism can be obtained by two main approaches, inter-query parallelism and intra-query Parallelism. Inter-query parallelism enables the parallel execution of multiple queries generated by concurrent transactions, and aims to increase the transactional throughput, while intra-query Parallelism aims to decrease response time. Intra-query parallelism could be composed by using ***inter-operator*** which is obtained by executing in parallel several operators of the query tree on several processors, and ***intra-operator,*** where the same operator is executed by many processors, each one working on a subset of the data.

In our experiment, a Parallel Query Interface (PQI) has been designed to facilitate the use of both of the approaches explained above, where a series of user's queries can be served at the same time. Each query can then be decomposed to sub-queries and executed concurrently, as explained in section 4.2

4 Experimental Environment and Results

In this experiment, Parallel Query Interface PQI was designed, and applied on the proposed Expandable Server Architecture. Different scenarios were produced (Central database on a single workstation with PQI and without PQI) in order to measures the performance of the query.

In this experiment, the TPC-D benchmark databases sets and their queries were used as well POSTGRES. The complexity of these queries are explained in [4]. POSTGRES [8] is an extended relational database system still being developed at Berkeley. It is well suited for handling massive amounts of data, it also supports large objects that allow attributes to span multiple pages and contains a generalized storage structure that supports huge capacity storage devices as tertiary memory [9].

The commercial parallel systems are very expensive, thus in this experiment a *Virtual Parallel Machine* (PVM) is used. This provides a cost-effective solution for small businesses.

In this experiment, a group of six workstations PIII 450 with 128 MB RAM, 20 GB hard disk linked with 100 MB/s network are connected via a LAN, which provides better cost/performance.

4.1 Expandable Server Architecture (ESA)

The fundamental objective of database technology is to retrieve the data in as short a time as possible.
This Architecture 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 all the workstations that are connected in the LAN and saving the small business, for example, from buying a multi processor server machine along with all its software tools. Using some or all those available resources in LAN, PVM is responsible for connecting them in order to perform any task in the best possible way. Performing any query in those selected resources in done using Parallel Query Interface (PQI) see figure 3, which uses a parallel mechanism to increase the performance and efficiency of retrieving the query. Also, by expanding the server, i.e., dividing up the database and spreading it over the LAN, we are speeding up the retrieval by accessing the data using the parallel mechanism for users' queries. Instead of loading up the main server and loosing time accordingly while retrieving any data.
This method reduces the communication overhead. As a result, there is a big performance benefit from reduced transaction latency and server workload.
Our system promises much higher flexibility in distribution of workload among nodes because any node can access the data at equal cost.

4.2 Parallel Query Interface (PQI)

The 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 [5]. 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. On the other hand, our ***Expandable Server Architecture ESA*** and ***Parallel Query Interface PQI*** (Coordinator Process) has all the relations allocated in different sites, we simply send each sub-query into the corresponding site. See figure 2. When a new user's query arrives, an arbitrary processing node receives it and becomes the coordinator in charge of optimization and supervision for this query. The coordinator first determines the degree of parallelism for the query by decomposing the query into sub-queries according to the join predicate and selection predicate. By determining the number of *Processing Nodes* (PNs) for scan and join as well as the number of disks that hold the buckets that are derived, and through message passing each operator can process the output of the previous one without delay.

The number of buckets is computed based on attribute value that has been retrieved from the corresponding disk and stored on the *table_info* at run time. When the buckets are processed separately, the coordinator will use the information from the *table_info* to command the PN, which holds the small buckets to sort them by

Quicksort algorithm. It will then send them to the according process to be joined, using a binary merge algorithm, according to the join predicate, see figure 1.
When the size of the buckets exceeds the main memory of any PN, a delay will be caused in the processing of the query, as can be observed from section 5. One way of avoiding disk contention, is to look at the inner and outer queries that are accessing the same data and use data fragmentation as in [7].

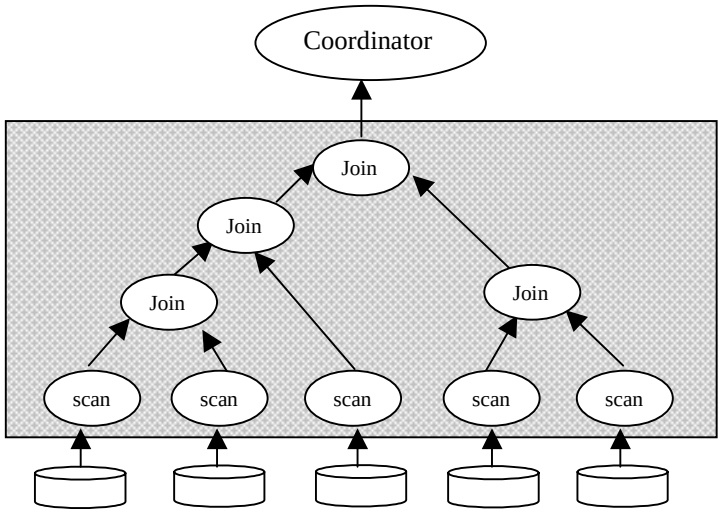

Figure. 1 Parallel Query Interface (PQI) applied in Expandable Server Architecture (ESA)

The *coordinator node* which applies the Parallel Query Interface PQI, executes the following:
Let Q :={Q_*Outer*, Q_*Inner}* // Q is the standard TPC-D benchmark queries, and contains:
// Q_*Outer* which is the outer sub-selection in the Q, and
// Q_*Inner*, which *is* the inner sub-selection in Q.
PN := { PN_0, PN_1, …PN_x} // it the processes Id which is being spawned
Let S_Q_*outer* := { O_1, $O_{2...}$,O_n } // is the result executing the outer query
Let S_Q_*inner* := { I_1, $I_{2...}$,I_m } // is the result executing the inner query
Let *AV* // Attribute Value
Let *table_info* // is the file where we save all the AV
Decompose the S_Q_*outer* and S_Q_*inner* // by: Gather all the predicates according to their
//table name for the S_Q_*outer and* S_Q_*inner.*
// join Predicates (compare two attribute), Predicates
//for select (value).
Spawn PNs to fetch each query in S_Q_*outer and* S_Q_*inner*
Keep track of the PN
Up date the *table_info* //by save the *AV*
The `coordinator` is in the listening status waiting for receiving PN form slaves' processes
DO {
If Receive PN_id_{outer}
Check *table_info* // to compare the size of the buckets
Find the smaller bucket // out of the buckets which based on the join predication
Command that PN to start sorting //check if its already sorted) their table
Send it correspond PN //using predicate join as counsellor.
Command the corresponding PN to start joining // using binary-merging according to
} //their predicate join}
Till we get Q_*outer* // it is the final join for the outer query

DO {
If Receive PN_id_{inner} {
Check *table_info* // to compare the size of the buckets
Find the smaller bucket // out of the buckets which based on the join predication
Command that PN to start sorting //check if its already sorted) their table
Send it correspond PN //using predicate join as counsellor.
Command the corresponding PN to start joining // using binary-merging according to their Predicate join }
}
Till we get Q_*inner* // it is the final join for the inner query
Finally join Q_outer with Q-inner //according to their predicate join

Figure. 2. Parallel Query Interface (PQI)

4.3 Experimental Results

In this experiment, query Q2 in the TPC-D benchmark, which is a correlated sub-query based on a 5-table join in both outer query and inner query [4], was performed in three different environments: ESA, Single workstation with imbedded Postgresql and without support of PQI and single workstation with support of PQI.

In order to effectively optimize queries in a distributed environment, it is necessary to have a reasonably accurate model that estimates the response time. Where 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 displayed at the user's site. As can be seen from figure 3, the elapsed time obtained from ESA is 88 sec. while at the workstation with support of PQI it was 177 sec. and at the workstation without support of PQI it was 236sec, see figure 3.

In the ESA environment, the sub-queries were executed in parallel, which saved considerable time in comparison with the other environments. In the case where one workstation was used with the support of PQI, the sub-queries were executed sequentially and the time of each execution was added up to give the overall response time of the main query. Finally, the case of one workstation without the support of PQI, the time was spent on the query process trying to find the best strategy plan for the execution of the query and on the sequential access of the data

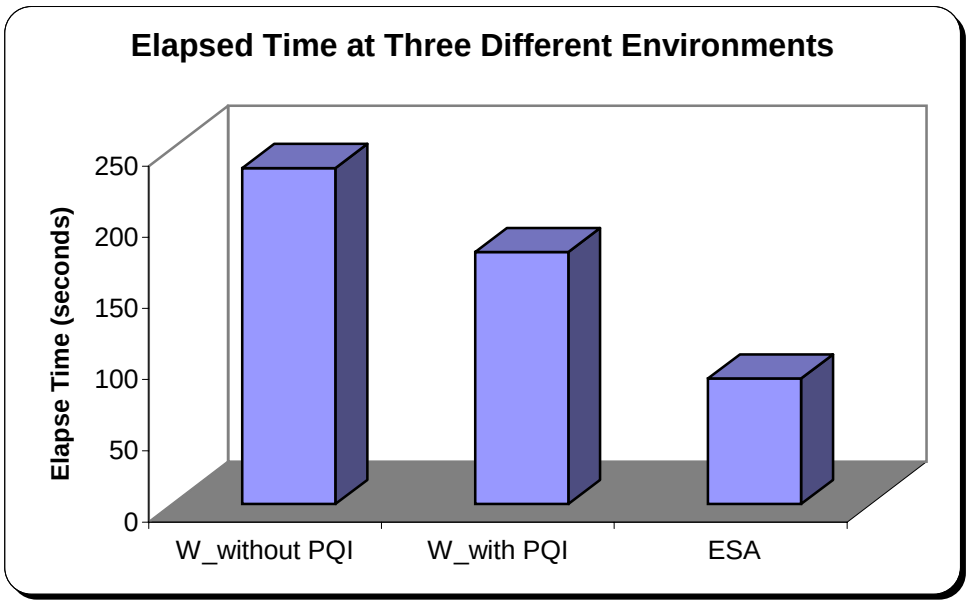

Figure. 3 Elapsed Time at three different environments; ESA, Workstation with support of PQI and Workstation without support of PQI

5 Parallel Virtual Machine (PVM)

The 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 nonblocking 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 nonblocking 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.

Message buffers are allocated dynamically. Therefore, the maximum message size that can be sent or received is limited only by the amount of available memory on a given host. There is only limited flow control built into PVM 3.3. PVM may give the user “a can't get memory” error when the sum of incoming messages exceeds the available memory, but PVM does not tell other tasks to stop sending to this host [6].
From figures 4, which represent the sending time for a range of data sets over different number of hosts, it shows that, the curves start off at a large sending time when the data is sent to one host and then the time reduces as the number of hosts increases. The curves eventually level off when the data are sent to about 5 hosts. The reason for this phenomenon, is that the size of the data when distributed over 5 machines fits totally in the buffer of the machines. This means time spent on page swapping when large sets of data are dealt with, is saved and there is less traffic to each machine. This complies with the way PVM communication model operates as explained above.

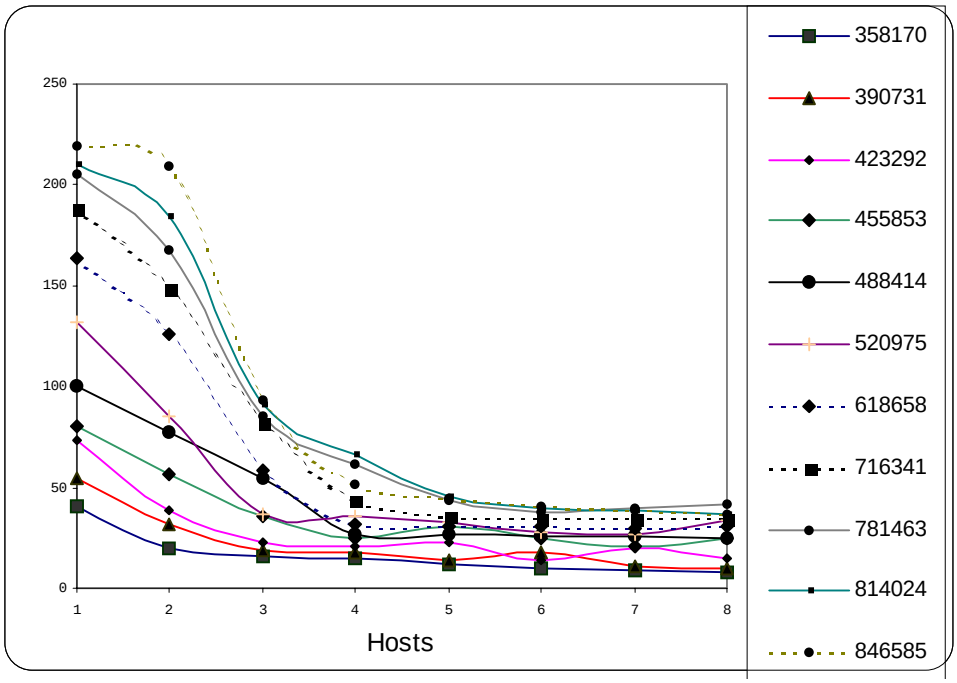

Figure 4 Sending time of 11 sets of data over 8 Hosts

6 Conclusion

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

Moreover, PVM shows significant improvement in performance by applying the proposed designed Parallel Query Interface PQI in various scenarios, namely Expandable Server Architecture ESA, a central database (in a single workstation) with support of PQI, and a central database (in a single workstation) without support of PQI. The experimental results clearly show that query performance seems to work best with ESA. The Expandable Server Architecture (ESA) uses the shared-nothing architecture that is popular for its scalability and reliability in connecting its processors. It is a cheap way of building a parallel database system that could be affordable to small businesses in contrast to the high cost of most commercially

available parallel database systems. The proposed PQI in this paper suggests several directions for future research, utilization of more workstations using PVM, enabling better load balancing and distributed partitioning.

References

1. 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
2. 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, 1997.
3. D. Dewitt and J. Gray.:Parallel Database Systems: The Future of High Performance Database System Comm. ACM, Vol. 35, No. 6 (June 1992) pp. 85-98
4. Carrie Ballinger.: Relevance of the TPC-D Benchmark Queries: The Questions You Ask Every Day. NCR Parallel System, www.tpc.org/articals/TPCDart1.97.html (1997)
5. M. Tamer Ozsu, Patrick Valduriez.: Principles of Distributed Database System. Second edition, ISBN 0-13-659707-6, Prentice Hall ,2000.
6. Geist et al, 1994 A Geist, A Beguelin, J Dongarra, W Jiang, R Manchek and V Sunderam (1994), 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
7. Michael J. Franklin, Bjorn Thor Jonsson and Donald Kossmann.: Performance Tradeoffs for Client-Server Query Processing, ,. SIGMOD Conf. 1996: 149-160
8. Michael Stonebraker and Greg Kemnitz.: The POSTGRES next generation database management system. Communications of ACM, 34 (1991)
9. Michael Allen Olson.: Extending the POSTGRES database system to manage tertiary storage. Master's thesis, University of California, Berkeley (1992)
10. Claire Mosher.: Postgres Reference Manual, version 4. Electronics Research Laboratory, University of California, Berkeley, CA-94720 (1992) No. UCB/ERL M92/85
11. S. Ganguly, A. Gerasoulis, and W. Wang.: Partitioning Pipelines with Communication Costs. In Proc. Of the 6th Intl. Conference on Information Systems and Data Management (CISMOD'95), Bombay, India (November 1995)
12. M. Mehta and D.J.DeWitt.: Managing Intra-operator Parallelism in Parallel Database Systems. Proc. 21st Intl. Conference on Very Large Data Bases, Zurich, Switzerland (September 1995)
13. A.N. Wilschut, J. Flokstra, and P. M. G. Apers.: Parallel Evaluation of Multi-join Queries. Proc. 1995 ACM SIGMOD Intl. Conference on Management of Data, San Jose, California (May 1995)
14. Rahm, E.: Dynamic Load Balancing in Parallel Database Systems. Proc. EURO-PAR 96, Lyon (1996), Springer-Verlag, Lecture Notes in Computer Science 1123, S.37-52, August 1996
15. Scheneider, D. A., DeWitt, D. J.: A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environmnet. Proc. ACM SIGMOD, Portland (1989).