Shared Memory and Shared

Databases

Computer System Architecture

Term Paper

Name: Ashwini Prabhu

Table of Content

Abstract

In today's data centric world we require the usage of shared databases for either integrating multiple applications so that they can work together or exchange information or in a data centric application used by multiple clients. But using a shared database has its own disadvantages. Mainly, we can hit performance bottlenecks that can't scale the desired load easily. Maintenance and development cost is increased. Administration becomes harder with difficult concurrency and version control.

Some of the disadvantages can be dealt with usage of Shared Memory. In today's world of virtualization and cheaper memory storage, we can use virtual data centers with shared memory. In this case multiple processes can access a whole block of memory simultaneously in a multiprocessing system. It deals with the performance bottlenecks by using shared Memory dedicated for databases and can also reduce the administrative overheads as version control is easier and we can use V2V data migrations while upgrading the databases which make Maintenance of the database easy.

1. Introduction:

Many applications like online transaction processing (OLTP) use a single database shared among multiple processes or sub-applications. When a large system consisting of many sub-programs needs the same database structure for all the sub-programs. Hence usually it ends up having a shared database. I was once working on was an online transaction processing that consisted of multiple modules sharing the same database. A single server was allocated for the database and all the applications were using the same database.

While working on that application it was found that there are a lot of advantages of this architecture. They are as following.

I. Advantages of shared databases [11]:

a. Ease of implementation.

b. Connecting processes together that require the same database schema.

c. Synchronizing data between processes and sub-processes.

d. Functionality encapsulated in one place

e. Quick and convenient data transfer because of immediate update feature.

f. No data duplication.

Although, shared databases have their own downside as follows.

II. Disadvantages of shared databases [12]:

a. Modifying the existing database structure is expensive because of tight coupling.

b. Cause high performance bottlenecks.

c. Difficult to define apparent boundaries between applications.

d. Expensive maintenance.

e. Lot of connections to the database at a time

This paper will shed light over the usage of shared memory to facilitate performance enhancements while working with shared database architecture. In shared memory architecture a single program creates a memory fragment that is made available to other processes with some synchronization mechanism. So multiple processes can access that memory segment without involving the operating system every time as control information is shared with all. The following diagram depicts the design of a shared memory system.

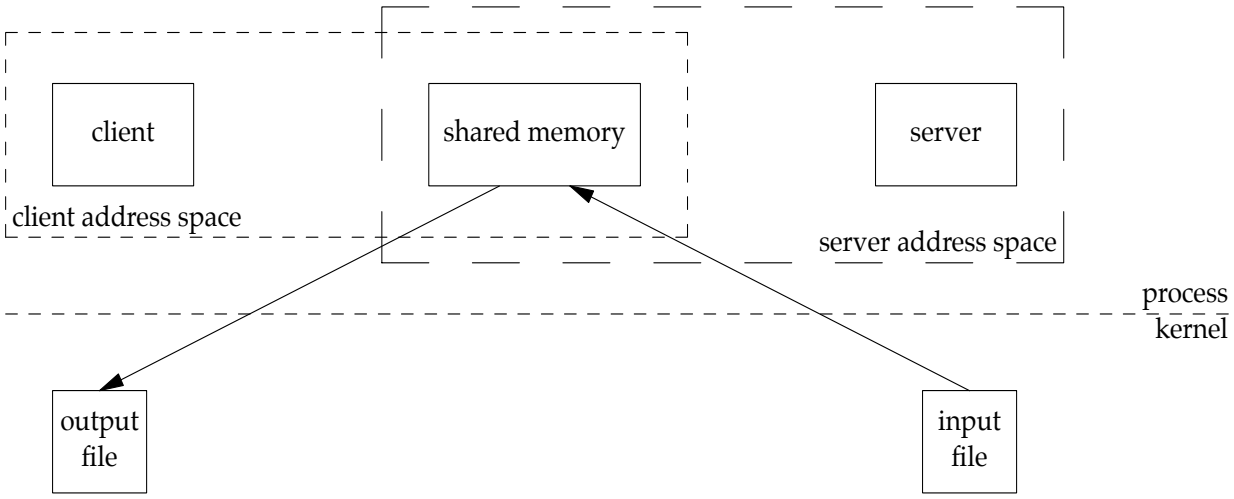

Fig. 1: I/O in shared memory [4]

It can be seen that the shared memory space is accessible to server as well as client and they can independently use it. Also the data is copied only twice, once from input file and once to the output file. Otherwise it would have been coped four times.

Also the involvement of the operating system is limited to I/O. The client and server can access the memory on their own.

2. Shared memory architectures [13]:

a. Uniform Memory Access: All the processes share the memory uniformly, i.e. access time is independent of the process requesting the memory.

b. Non-uniform Memory Access: Access time depends on the memory location with respect to the processors.

c. Cache-only Memory Access: Local memory used as cache instead of using as main memory.

Shared-memory has many advantages of shared-database architecture. Some of it's own advantages are as follows.

I. Advantages of shared memory [14]:

a. Much faster processing because of OS-independent functions.

b. Reduced risk of data duplication

c. Less wastage of resources

d. Simplicity of usage

e. Strong load balancing

3. Shared-databases using shared-memory:

In this architecture the memory is in between the processes and database. To access the database the processors have to access the memory first. Any fragment of the memory can be accessed through the interconnect bus which is very fast. In this case physical as

well as virtual memory is shared. The load balancing in this kind of architecture depends on the partitioning of the database on which the database application workload is tested.

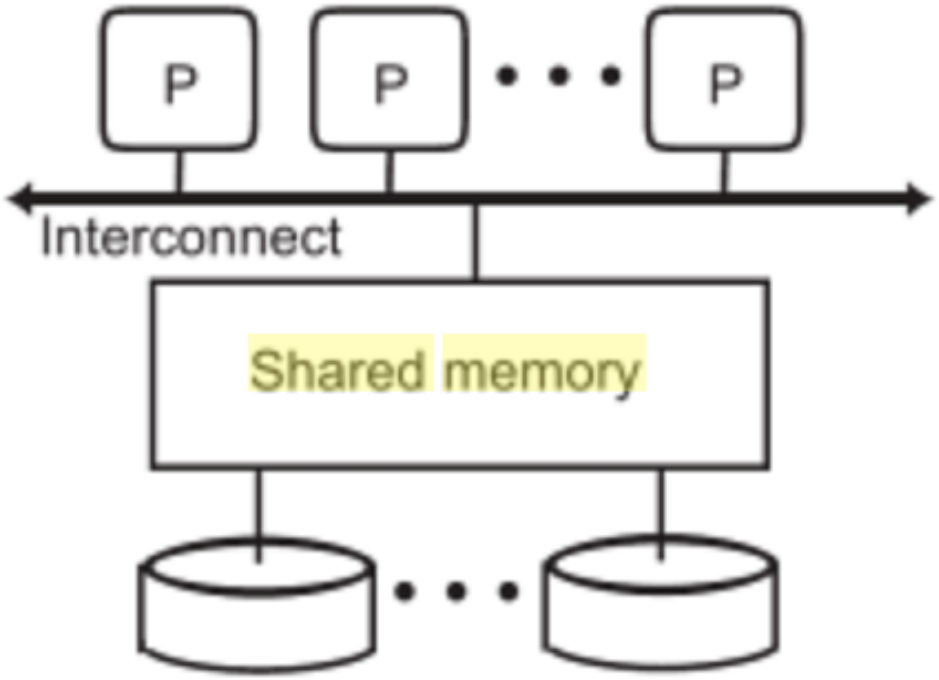

Fig. 2: Shared memory architecture [5]

A lot of experiments have been conducted to test the performance of databases using shared memory.

I. Experiment 1 and observations

One of them was performed using Non-uniform Memory Access (NUMA). As seen before in NUMA systems the cost of accessing the memory may vary depending on the location of the memory. The system consisted of 4 processors sharing the memory. [1]

A. Database testing applications:-

For the testing purposes the commercial edition of Oracle's database management system was used. The server programs run the database transactions. A few threads are created to perform other duties like writing the logs or to the database. For communication client processes use a pipe and other processes use a shared memory region consisting of buffer area for actual data and metadata area. [1]

B. Online transaction application:-

It's a banking system that takes cares of account balances of the customers depending on the branch where the account is. Transactions modify a random account balance. Along with the account balance the transaction also has to update the balance of the customer's branch. The database also contains a history table for all submitted transactions that needs to be modified after the transaction is completed. [1]

Every client process has a separate process on the oracle server for running the transactions.

C. Decision support system application:-

To keep it relatively distributed over all the big tables the used query has correlative entities to keep track of the attributes that are majorly used. By using the appropriate running mode the query was divided into parts and a different server process ran each part. [1]

D. Method of testing:-

Along with the programs ran to test the system the synchronization information is also maintained. The information used in locking is also maintained. It helps in obtaining

synchronization in simulated processes. It was observed that every process was executing independently of other processes. In total 200,000,000 instructions were executed to test the performance. [1]

E. Database scaling:

Every processor was given the same number of tasks to run. Database had 40 branches and the size of shared memory was more than 900 megabytes. For online transaction processing eight processes were assigned to each processor and for decision support system. [1]

F. Architecture of the system:-

Each processor of the four had its own level 1 and level 2 caches, access to the shared memory, interface component for network, directory. A bus connects all these components together.

The level one cache has two ports and uses write allocate write back policy. The level two cache is completely pipelined and it also uses write allocate write back system. All caches are addressed using physical locations. Registers are provided to provided to hold the information about the status of the cache.

For execution purposes a special branch prediction policy was used for the branching conditions, a branch target buffer for jumping branches and a stack for branches returning back to the calling statement. [1]

G. Performance:-

From the observations of the tests run it can be seen that as the window size for instructions was increased the execution time went on decreasing. For stall cycles, ratio

of the instruction exiting a particular cycle to the maximum exit rate was calculated and this fraction was referred to the busy time. The other part of the fraction is referred as stall time for the first instruction that was not able to exit the cycle. [1]

There were separate conventions for memory instructions. A few extra cycles can be referred to the instruction plus the time spent in execution.

The time spent in execution can be divided into two parts. First while generating address it takes time because of data dependency and second when the pipeline needs to be restarted after misinterpretation of branch, cache miss or memory instruction is at the head of the window. Even though the stalls are there for every memory type they are evident in level one cache. It can be seen from the results that the level-two cache component belongs to the online transaction processing workload because of the large instruction impression. [1]

When the window size is increased the performance is found to be improving. The level-two cache fragment plays a major role in this improvement. It can also be seen that the performance is bounded by the characteristics of the calculation.

Moreover the performance improvement for decision support system is better than the online transaction processing. It is because of the accelerated computing nature of decision support systems that shortens the stall time. [6]

H. Limitations:

Even though the instruction level modifications improve the system performance, there are other factors that affect it more than anything. Issues like data miss and instruction miss can lead to considerable amount of lag in the performance. Some steps have been taken to improve things on that front. Correct branch predictions and a better

cache for instructions improved the performance considerably. Addition of increased window size to the given configuration decreased the execution time a lot. It was observed that higher instruction misses block branch prediction system from improving the performance to a measurable level. But the availability of infinite cache for instructions provides a greater effect, which reduces the instruction stall time. [1] Additionally increasing the functional elements the performance increases. When compared with systems having a single processor, it can be seen that for online transaction processing the stall time for instruction is lesser as there are n data misses.

There are some other techniques that are used to boost the performance. Hardware prefetching is used to issue nonessential pre-fetch for operations with known addresses but are unavailable for some reason. In the other technique hardware support is required to detect violations and recover from them. [7]

The performance is also tested with consistency models. It can be seen that release consistency has the greatest performance improvement. Processor consistency is at the second level with moderate performance improvement. [8] Sequential consistency shows low performance improvement than the other two. Optimization has a great effect on the performance improvement in case of processor and sequential consistency. Release consistency shows a lesser effect of optimization. Prefetching the instructions has a considerable advantage because the computations are dependent on the data. Along with speculative load the effect of prefetching is much more than expected. From past studies it was found that there is some difference in performance with sequential consistency and release consistency. But nowadays microprocessors are available with

some optimization processes done on them. Hence the consistency models may not have as much effect as seen before. [1]

Hence it was concluded that the benefits for decision support system are much more important than the benefits for online transaction process. Online transaction processing system has a bigger memory system fragment that is difficult to ignore. Also the processors used in the experiment grant optimized usage of consistency models that Considerably improve the performance of the tougher models.

I. Instruction addressing and bottlenecks:

Level-one cache misses that hit in the level 2 cache controlled the stall time for instructions. A larger fragment of instruction references pursues a streaming pattern with consecutive references to contiguous positions in the address space. It may be possible that having an extra buffer for instruction streaming between level 1 and level 2 caches improves the performance.

Stream buffer is used to prefetch to consecutive cache lines following misses. It's a hardware based prefetching technique. To have a cleaner cache the prefetching signals are stored in the instruction stream buffer till the prefetch data is used. If a cache miss does not match with any of the values in the buffer the hardware discards all the values present in the buffer and starts prefetch again. An instruction stream buffer is very useful for instruction misses in online transaction processing. It has been observed that a 2 unit instruction stream buffer can have reduction rate of almost 64%. The misses in the instruction stream buffer reduced as streams in the online transaction processing system were not very long and some extra entries in the stream buffer can affect the performance due to unwanted prefetches in the level 2 cache. [1]

In the experiment it was observed that as we increase the unit number in the stream buffer the execution time goes on decreasing and hence performance increases. Most of the performance gain was obtained by the overlapping of instruction misses in the level 1 cache. When analyzed further, it was found that a bigger component of the instruction misses were repeating the patterns with an irregular pace. One way to tackle these misses would be the realignment of the instructions by an analyzing program such as compiler that connects with the branching buffer to execute prefeteches for the correct branching option. [1] But it is possible that these improvements are limited by the precision of the branch prediction algorithm and the hardware cost. [9] Instead of using an instruction stream buffer is to increase the transfer element between level 1 and level 2 cache. Although it was seen that stream buffers have the capacity to get used to longer stream lengths on the fly without increasing the accessing time. Hence including an instruction stream buffer to the architecture a great performance improvement can be achieved for online transaction processing with low cost.

J. Data miss bottlenecks:

The studies have shown that cache read misses in the level 2 cache add to some performance related issues. One because the shared write access and read cache misses are related to the data that reflects a sharing pattern. In online transaction processing there are a lot of small changes going on all the time. So the data structures are passed along with the locks in the synchronization process. Also the moving misses are controlled by a small part of the passed data.

There are two solutions to reduce the performance degradation. One is a program that can characterize requests for those data structures and can do a prefetch of the data.

Current CPUs have the required means to execute such programs. Second is a program that can find where the last instruction for accessing the passed data resides, can empty the memory or update it with the current values. Hence the memory can take care of the next data read instructions and reduce the time taken by cache to transfer the data. To empty the memory effectively it is necessary to have a clean data in it otherwise the latency of next data misses will balance the performance increase. [1]

Hence for the testing purpose prefetch and memory emptying code was included later. In short a 4-unit stream buffer was used for optimization. Adding the properties for emptying the memory at the appropriate position increases the performance a lot by reducing the execution time. Mainly the time taken by read misses in the memory is reduced considerably. Considering the impact of flushing code together with stream buffer the performance improvement is promising. Unfortunately late prefetches and contention issues seem to prohibit the impact of the two factors discussed previously. [1]

II. Experiment 2 and observations -

Another such experiment done on shared memory for databases shows that it's a very useful architecture. It shows that this kind of architecture shows great amount of compatibility and transparency and are very easy to implement. The database systems store the data in the physical memory so that it can be accessed quickly. Databases can be designed with shared memory to achieve performance improvement and correct results. Although some performance degradation may occur where systems don't have a better processor locality and situations with false sharing can take place, so the hardware component picks up the tasks of the software. To avoid it a solution is to have relaxed

coherency, which might not be a convincing option. Hence this experiment is performed with a system that has a strict coherency approach.

A. Experiment details:

This approach is very cheap and has an advantage of providing quick access to memory to the whole network. Also in shared memory architecture the application do not have to worry about the protocols present at the lower level of the architecture, which is an alternative to passing messages. So for performance improvement it is suggested that the coherence level should be low, so the system can perform better and many programs can work well together. [3]

But there are some techniques that do not require any kind of coherence configuration. They can work well in case of strict coherence too. So the processes requiring benefits of coherence can also go on. Databases need shared memory to store persistent data reliability. It was observed that it might be possible for a few reliable, well-behaving, communicating sub-systems. Failure modes in some systems have site failure and network partitions. It was found that reliability support is very critical to save the integrity of the data stored in the databases. [3]

B. Performance improvement techniques:

One of the techniques is time-based coherence that tries to reduce the number of page faults created and the expense related to strict coherence maintenance. That site uses a time window to get a fixed time span that works on a particular site may possess that part of the shared memory asked. This time window tries to provide some level of fairness between the system using that part of memory and the systems that are

requesting the memory fragment. The time window provides some level of control on the amount processor locality, which is the number of processors attribute to a given memory fragment before some other processor is allowed to refer. [3]

It was seen that this technique works fine especially with applications thrashing pages between processes. In this process the time window provides an implicit locking system that prohibits fragments of the shared memory being ranked lower. [6] In absence of this time window the part of the application that could be written on would be degraded between steps, which would take place without any interruptions. It is not possible to completely remove premature denial of an application fragment though the removal of some amount of premature page denial may add to the performance improvement considerably. [3]

By using the time window it was observed that the simulation was over in nearly half of the time taken by the version without the time window or time-based coherence. It is helpful in other situations also such as places where processes reside at the same location and share the data. Most of the times the resources for context switching and giving data access can be divided into the time window so that every process gets equal share in everything. [8]

C. Compression:

It is a very critical factor in shared memory systems as it can reduce the degree of data transferred over the network considerably. The observations reveal that the performance of the system improves radically by using compression as it could reduce the amount of network interrupts since it will require to process less number of data packets. Additionally compression allows systems with shared memory to scale up to a

bigger system set up as the amount of network transfer among processes reduces substantially and the network traffic reduces as well. [3]

Even though the observations encourage the compression technique so well, the techniques used in shared memory systems have some constraints that are different than regular systems like software overhead or latency. Compression parameter should have the exact required value. Compressing to any intermediate proportion will not decrease the network transmission that needs to be optimized for better performance improvement. This affects the performance significantly. In some situations some more time spent in compression for greater compression effects may be taken care of by decreasing the network packets. In the opposite case compressing data more over the border will have little effect, as it does not reduce the number of packets that are processed. [7]

Hence it was concluded that time-based coherency and compression can have a great impact on performance in the shared memory systems. It is possible to improve the performance significantly without having to relax coherence. Also it might be necessary for some applications to have strict coherence.

D. Reliability of the system:

Reliability is an important factor in databases based on shared memory, as these databases need to store very persistent data. [6] The issue of site failures needs to be considered on all levels, which are very common to occur. It is very critical since a considerable amount of non-sequential writable fragments could be lost because of the fault. [10] Even though there are techniques like duplication or replication and multicast, not all systems can afford to use them.

The issue of failure requires being resolved quickly since it can leave blanks in the address space very easily. In a strictly correlated process it may or may not be possible to recover from the faults. But if provided with the recovery program then the application must be able to invoke it in case of failure. That can create another process, which can take over the remaining part of the failed process and complete the task successfully. For the experiment a protocol was devised that was used for maintaining a page at the location of the process, which previously accessed the data to recover from the failure and continue with the failed process. [3]

But maintaining the failed pages is not what is enough for this purpose. For example if a process is going to fail in some time where that process has read data from a shared memory and has updated another memory location. Then that data is deleted from location and given to some other process and then that process fails. If we try to redo the steps as they occurred then some other process may repeat those updated if they have been recorded. [3] Locking provides a useful means of recording the updates in databases. So consistency can be maintained by implicit as well as explicit locking. [8]

The issue of consistency can also be resolved by using time-based coherency to maintain groups of memory fragments in a given process waiting for a commit operation to be performed. The time window can be used to accomplish consistent updates happening at a particular stage in all the processes in the network. For that purpose the memory segments being used in a particular process must be maintained and updated in a group. Hence after a specific amount of time updates to the memory segments will be provided to the other processes for commit. When the updates are committed the new nature is enduring to failures and hence either all the pages or none of them must be

available in the network. But this technique affects performance considerably because of some extra overheads. [3]

III. Experiment 3 and observations -

Another experiment done was on scalability factor of the shared memory architecture. There are two factors that restrict scalability a great deal. First is related to the algorithms used in the program and comes into picture because of imbalance in the processes and fraction. The second is because of the communication between the program algorithm and the architecture and arises because of latency time and hassle in the network. A top down approach can work great in case of shared memory.

A. Introduction -

The overhead functions related to different characteristics of architecture and the software are defined to evaluate the scalability of the system. A system can be developed to divide the overhead related to the software into a serial component and work imbalance component. Similarly a process is created to detach the latency overhead from the total time taken by the application for execution.

Scalability usually determines the quality of architecture. It provides the guidelines to configure the perfect architecture for a specific system, determine the performance of it on a bigger configuration with an existing system, identify the limitations to the application, identify the bottlenecks that might get created because of the architecture or program and the communication between the architecture and the application program that can tell the steps taken for scalability. [2]

The main point of using top down approach is to find out the major factors of architecture and the programs hampering the performance improvement, get to know the communication between them calculate the impact of these factors on the time required for the application to execute. [2]

Some measures have been taken to improve the scalability of the architecture that are speedup, scaled speedup, serial fraction determination and sizeup. Even though these factors provide proper boost in the performance improvement they are unable to improve the scalability of a system. [2]

For this purpose some performance studies have been taken into consideration that are related with the issues like latency and synchronization. These factors play an important role in the performance improvement issue hence it is better to consider them in the overall performance evaluation. But not a lot is known about the effects of the interaction between architecture and algorithm.

B. Experiment details:

It is expected that when number of processors in a system increases the performance improves along with it. But it is observed that there are some overheads related to it, which hamper the performance. Even if some more power is added to the system, after a specific processing stage the overheads tend to affect the performance. There are two types of overheads, first is related to the algorithm and the second one is related to the interaction between the architecture and the algorithm. Making these two parts separated can be done to reconstruct performance-improving code. [2]

It is necessary to isolate the impact of each interaction participant on the total performance to find out the scalability of a system. The important features of the

experimental approach are usage of real world applications, finding the participating processors, analyze the interaction of components and anticipate the scalability of the system under experiment.

Scalability studies are a little difficult because there are a lot of factors that can be changed to get better and better results. Experimentation is necessary in the performance scalability but there are a few shortcomings. [10] First is that the architecture of the system is unchangeable hence it is not possible to analyze the impact of the changing characteristics of the hardware. Another is that it is very difficult to isolate the individual effects of the architectural parameters on the total system performance. Additionally analyzing the system behavior using instrumentation can give wrong results. Analytical models used to provide overall performance estimates for large systems tend to make assumptions about behavior of the process and the hardware characteristics to analyze

The performance. Its not easy to control the overheads using a few factors that's why the simulations are done. But those are taken care of to some extent in this experiment. The overhead caused by the algorithm is evaluated by calculating the time required for execution of the program on an ideal machine. Processes sharing memory need to communicate while working. It is done through modifying the shared variables. Although in any kind of communication the processes need to access the network and the physical properties can have some impact on the execution overheads, which are unavoidable. [2]

One of the overheads is latency, which is calculated as the total time a processor spends while waiting for a message because of the time taken by the network to pass the message and the switching time required for the system. The other overhead is

contention, which is calculated as the overall time induced by a processor while waiting for free resources. It shows that communication is not connected to synchronization in a shared memory system. They either have some basic synchronization procedure or some hardware support for complex operations.

This saves a lot of execution time but its not very flexible. Hence if we neglect these synchronization operations and the overhead of them then we have to care only about the overhead generated by message going in two ways and this overhead is considered in latency and contention. The waiting time taken by the processors while synchronization is included in the total time as algorithmic overhead. In the studies of such systems there are three types of scales that are used. Constant problem size, memory constrained and time constrained. Overheads can be easily calculated using any of the three scales. For this experiment the scaling made used is constant problem size. [2]

The parameters that were continuously on the check were number of processors, clock speed, time required for the hardware to transfer a message, software latency and hardware and software bandwidths. The application provides a lot of statistics about the program. The total time is the maximum of all the execution times of different processors. The evaluation of latency and is done by time-stamping messages when they are sent. When a message is received the time taken by a message for transmission is considered as latency overhead and remaining time is counted as contention overhead. When latency and contention is observed when receiving a message the processor can nullify its effects by calculations and communication. [2]

C. Architectural Details:

For the experiment three shared memory platforms were used with three different topologies, fully connected, binary hypercube and 2-d mesh. All of them used unidirectional serial links. The fully connected model has links between each processor pair. In the binary hypercube platform each edge of the cube has a link in all the directions. In the 2-d mesh architecture a processor has links going to all the other neighboring processors in the network. The messages are transferred using circuit-switching technique. It sets up a network between the source and the destination upon the beginning and a message is sent as a packet. The circuit doesn't take much time to set up. This architecture is a cache coherent non-uniform memory access architecture. Each processor has access to a large size of shared memory such that the data set given to each memory can reside in the portion of the shared memory assigned to each processor. Also they are provided with a 2 way private cache that is maintained using a very complicated cache coherence scheme. [2]

The results of the simulation were clear enough to show how the latency and contention overheads were growing as compared to the number of processes and their effect on scalability. All the processors had large caches hence it can be said that the latency overhead would have occurred only due to the processor. It can also be said that because of the irregularities in the synchronization process the overheads may differ depending on a network and its parameters. But it was observed that contention overhead increased as the connectivity went on decreasing in the network. [2]

Considering individual processors the computation time required for a processor is more than the time required for communication. Also the contention in a network fully connected is negligible because of message queuing. But with lesser connectivity the

contention increases as the number of processes goes on increasing. It was observed that the constant factors related to the execution time were longer than that of latency and contention. Even though increase the number of processing statements increases the values of factors related to execution time, other factors are unaffected. [2]

The experiment was done on an execution driven simulation platform. It was possible to separate the overheads caused by the algorithmic characteristics and by the interaction between the architecture and the algorithm. Because of that it was possible to closely analyze the factors related to the architecture and the design that could affect the scalability of system considerably. Overheads related to the algorithm like some extra processing can limit the scalability as well. When shared memory systems with local caches are into consideration, it is observed that in case of well constructed programs the main contributing factor to affect scalability is the delay in the network. This has been observed for most of the shared memory systems available till date. [2]

4. Conclusion:

The need of shared resources is increasing. A lot of systems have been brought to the market that provide the required services. To design a well suited system for the required purpose it is necessary to test the architecture under experimentation for all the required factors. The experiments considered above provide detailed study of the characteristics of the architecture and as well as the software design. They are useful in understanding the flaws in the design and make some improvements in online transaction processing. The shortcomings of that can be nullified by techniques like instruction stream buffer, prefetch and flush operation. [10]

Additionally the performance can be improved further by isolating the algorithm design component from the architectural view. It is easy to reduce the overhead generated by separately studying these two overheads and figuring ways to eliminate the factors causing them and improving the scalability of the system. It might also be helpful in getting rid of the additional work contributing to the decreasing scalability. Also it is important for the applications to be well structured for better performance.

5. References:

[1] ACM Journal: “Performance of database workloads on shared-memory systems with out-of-order processors” by Ranganathan ,Parthasarathi; Gharachorloo , Kourosh; Adve , Sarita V. ; Barroso, Luiz Andre`, (1998). Proceedings of the eighth international conference on Architectural support for programming languages and operating systems, Pages 301-318.

[2] ACM Journal: “An approach to scalability study of shared memory parallel systems” by Sivasubramaniam, Anand; Ramachandran, Umakishore; Singla, Aman; Venkateswaran, H., (1994) SIGMETRICS '94 Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems, Pages 171-180.

[3] ACM Journal: “Memory consistency and event ordering in scalable shared-memory multiprocessors” Kourosh Gharacharloo, Daniel Lenoski, James Laudon, Philip Gibbons, Anoop Gupta, John Hennessy (1990). ISCA '90 Proceedings of the 17th annual international symposium on Computer Architecture, Pages 15-26

[4] Online source: “http://www.kohala.com/start/unpv22e/unpv22e.chap12.pdf”

[5] Reading Material: “Principles of Distributed Database Systems” by M. Tamer Ozsu, Patrick Valduriez.

[6] IEEE tutorial: “Shared memory consistency models” by Sarita V. Adve, Kourosh Gharacharloo, IEEE Computer Society, vol. 29, issue 12, p. 66-76, August 2002.

[7] IEEE journal: “shared memory computing on networks of workstations” by Amza, C.; Cox, A. L.; Dwarkadas, S.; Keleher, P.; Honghui Lu; Rajamony, R.; Yu. W.; Zwaenepoel, W., IEEE Computer Society, vol. 29, issue 2, p. 18-28, August 2002.

[8] IEEE Journal: "SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery" by Sorin, D. J.; Martin, M.M.K.; Hill, M.D.; Wood, D. A., Computer Architecture, 2002. Proceedings. 29th Annual International Symposium, p. 123-134, May 2002.

[9] IEEE Journal: "Algorithms implementing distributed shared memory" by Stumm, M.; Zhou, S.; IEEE Computer Society, vol. 23, issue 5, p. 54-64

[10] ACM Journal: "Algorithms for scalable synchronization on shared-memory multiprocessors" John M. Mellor-Crummey, Michael L. Scott, ACM Transactions on Computer Systems (TOCS), vol, 9, issue 1, Feb 1991.

[11] Online source: "http://www.ben-morris.com/a-shared-database-is-still-an-anti-pattern-no-matter-what-the-justification/"

[12] Online Source: "http://en.wikipedia.org/wiki/Shared_memory_architecture"

[13] Online Source: "http://www.cs.cf.ac.uk/Dave/C/node27.html"

[14] Online Source: "http://www.kohala.com/start/unpv22e/unpv22e.chap12.pdf"

[15] Reading Material: "Readings in Database Systems" by Joseph M. Hellerstein, Michael Stonebraker, Fourth Edition, 2005.