Chapter 4. **Parallel database**

4.1. Introduction to Parallel Database:

- A **parallel database** system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries.

  Parallelization:
  1. **Parallel computing** is a type of computation in which many calculations or the execution of processes are carried out simultaneously.
  2. Large problems can often be divided into smaller ones, which can then be solved at the same time.
  3. There are several different forms of parallel computing: bit-level, instruction-level, data, and task parallelism.

- Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations.
- Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel.
- Centralized and client–server database systems are not powerful enough to handle such applications.
- In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.

Advantages:

1. Performance Improvement –

By connecting multiple resources like CPU and disks in parallel we can significantly increase the performance of the system.

2. High availability –

In the parallel database, nodes have less contact with each other, so the failure of one node doesn't cause for failure of the entire system. This amounts to significantly higher database availability.

3. Proper resource utilization –

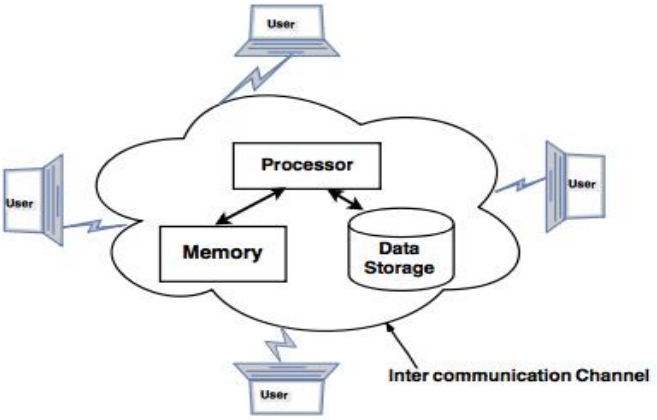
Due to parallel execution, the CPU will never be ideal. Thus, proper utilization of resources is there.

4. Increase Reliability –

When one site fails, the execution can continue with another available site which is having a copy of data. Making the system more reliable.

5. Provide distributed access of data

Companies having many branches in multiple cities can access data with the help of parallel database system.



Parallel database system

5.2. Parallel Database Architectures

1. **Shared memory architecture:**

   - Where multiple processors share the main memory (RAM) space but each processor has its own disk.
   - If many processes run simultaneously, the speed is reduced, the same as a computer when many parallel tasks run and the computer slows down.

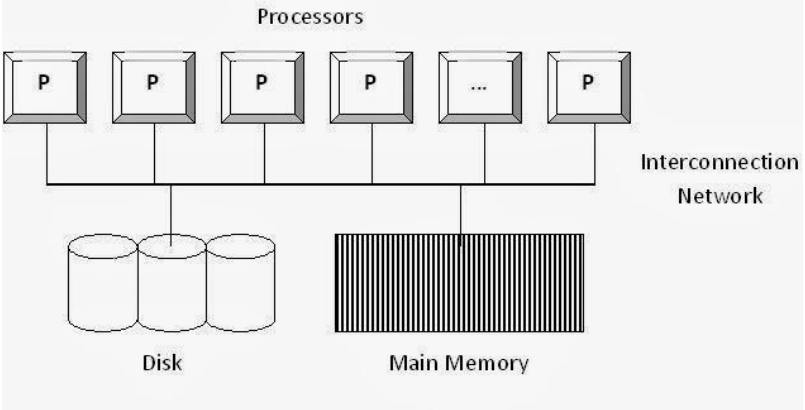


Figure 1 - Shared Memory Architecture

In Shared Memory architecture, single memory is shared among many processors as show in Figure 1. As shown in the figure, several processors are connected through an interconnection network with Main memory and disk setup. Here interconnection network is usually a high-speed network (may be Bus, Mesh, or Hypercube) which makes data sharing (transporting) easy among the various components (Processor, Memory, and Disk).

Advantages:

- Simple implementation
- Establishes effective communication between processors through single memory addresses space.
- Above point leads to less communication overhead.

Disadvantages:

- Higher degree of parallelism (more number of concurrent operations in different processors) cannot be achieved due to the reason that all the processors share the same interconnection network to connect with memory. This causes Bottleneck in interconnection network (Interference), especially in the case of Bus interconnection network.
- Addition of processor would slow down the existing processors.
- Cache-coherency should be maintained. That is, if any processor tries to read the data used or modified by other processors, then we need to ensure that the data is of latest version.
- Degree of Parallelism is limited. More number of parallel processes might degrade the performance.

2. **<u>Shared disk architecture</u>:**

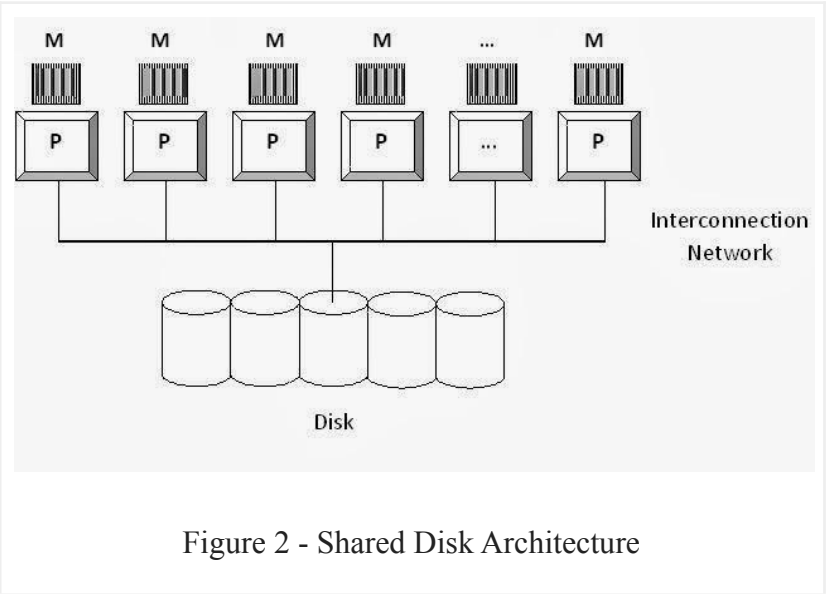


Figure 2 - Shared Disk Architecture

In Shared Disk architecture, single disk or single disk setup is shared among all the available processors and also all the processors have their own private memories as shown in Figure 2.

Advantages:

- Failure of any processors would not stop the entire system (Fault tolerance)
- Interconnection to the memory is not a bottleneck. (It was bottleneck in Shared Memory architecture)
- Support larger number of processors (when compared to Shared Memory architecture)

Disadvantages:

- Interconnection to the disk is bottleneck as all processors share common disk setup.
- Inter-processor communication is slow. The reason is, all the processors have their own memory. Hence, the communication between processors needs reading of data from other processors' memory which needs additional software support.

Example Real Time-Shared Disk Implementation

- DEC clusters (VMScluster) running Rdb

3. Shared nothing architecture

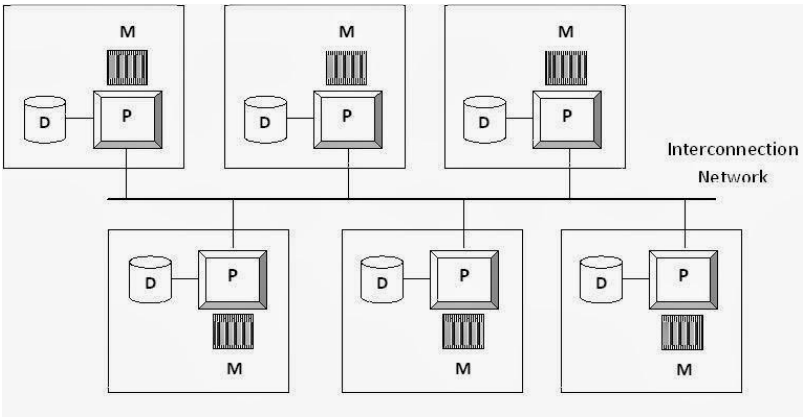


Figure 3 - Shared Nothing Architecture

In Shared Nothing architecture, every processor has its own memory and disk setup. This setup may be considered as set of individual computers connected through high speed interconnection network using regular network protocols and switches for example to share data between computers. (This architecture is used in the Distributed Database System). In Shared Nothing parallel database system implementation, we insist the use of similar nodes that are Homogenous systems. (In distributed database System we may use Heterogeneous nodes)

Advantages:

- Number of processors used here is scalable. That is, the design is flexible to add more number of computers.
- Unlike in other two architectures, only the data request which cannot be answered by local processors need to be forwarded through interconnection network.

Disadvantages:

- Non-local disk accesses are costly. That is, if one server receives the request. If the required data not available, it must be routed to the server where the data is available. It is slightly complex.
- Communication cost involved in transporting data among computers.

Example Real Time-Shared Nothing Implementation

- Teradata

- Tandem
- Oracle nCUBE

4.3 I/O Parallelism

- **Parallel I/O**, in the context of a computer, means the performance of multiple input/output operations at the same time, for instance simultaneously outputs to storage devices and display devices. It is a fundamental feature of operating systems.
- One particular instance is parallel writing of data to disk; when file data is spread across multiple disks, for example in a RAID array, one can store multiple parts of the data at the same time, thereby achieving higher write speeds than with a single device.
- Reduce the time required to retrieve relations from disk by partitioning the relations on multiple disks.
- **Horizontal partitioning** – tuples of a relation are divided among many disks such that each tuple resides on one disk.

- **Partitioning techniques** (number of disks = n):

1) **Round-robin**: Send the i^{th} tuple inserted in the relation to disk i mod n.

Round robin method always creates approximately equal size partitions. The first record goes to the first processing node, the second to the second processing node, and so on.

For example, if you have 3 nodes in your database partition.

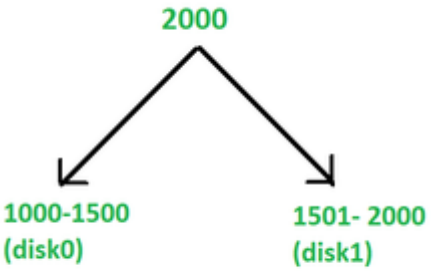
1st ROW will go to node1
2nd ROW will go to node2
3rd ROW will go to node3
4th ROW will go to node1
5th ROW will go to node2
... And the process keeps on rotating

2) Range partitioning

Rows in a range-partitioned table or index are distributed among partitions according to values in the partitioning key columns. The partitioning column values of each row are compared with a set of upper and lower bounds to determine the partition to which the row belongs.

- Every partition has an inclusive upper bound, which is specified by the **values <=** clause when the partition is created.
- Every partition except the first has a non-inclusive lower bound, which is specified implicitly by the **values <=** clause on the next-lower partition.
  - For example, we have 3 disks numbered 0, 1, and 2 in range partitioning, and may assign relation with a value that is less than 5 to disk0, values between 5-40 to disk1, and values that are greater than 40 to disk2. It has some advantages, like it involves placing shuffles containing attribute values that fall within a certain range on the disk.

See figure 1: Range partitioning given below:



3) Hash partitioning: –

In hash partitioning, Adaptive Server uses a hash function to specify the partition assignment for each row. You select the partitioning key columns, but Adaptive Server chooses the hash function that controls the partition assignment.

- This declustering strategy designates one or more attributes from the given relation's schema as the partitioning attributes. A hash function is chosen whose range is {0, 1, . . . , n − 1}. Each tuple of the original relation is hashed on the partitioning attributes. If the hash function returns i, then the tuple is placed on disk Di.

Hash partitioning is a good choice for:

- Large tables with many partitions, particularly in decision-support environments
- Efficient equality searches on hash key columns
- Data with no particular order, for example, alphanumeric product code keys

If you choose an appropriate partition key, hash partitioning distributes data evenly across all partitions. However, if you choose an inappropriate key—for example, a key that has the same value for many rows—the result may be an unbalanced distribution of rows among the partitions.