Topic 5.3: Deploying Databases on Cluster Servers

As data size and processing requirements exploded, businesses sought ever more powerful computer systems to handle the processing loads. Moreover, businesses started demanding that computer systems remain up and available anywhere from 99% to 99.999% of the time. To put those numbers in perspective, 99% "uptime" allows for just 12 minutes of downtime in a 30-day period, while 99.999% uptime allows for just 26 seconds of downtime in a 30-day period. Technicians quickly realized that these performance and reliability requirements could not be met with single computer systems. First, redundant components (network connections, power supplies, memory, CPUs, disks, etc.) are required to eliminate all the "single points of failure" that can bring down a computer. Second, there are limits to how fast electrons can move through circuits. The obvious solution was to start combining computer systems (typically servers) together so that they could break larger tasks into smaller ones that could be processed in parallel, while also providing fault tolerance such that when one server fails, the others can pick up its work and continue without interruption. Interestingly enough, the earliest such systems (in the early 1960s) were not developed by vendors, but by their customers -- out of pure necessity. Vendor-designed solutions began to appear some years later.

Cluster Server Architecture

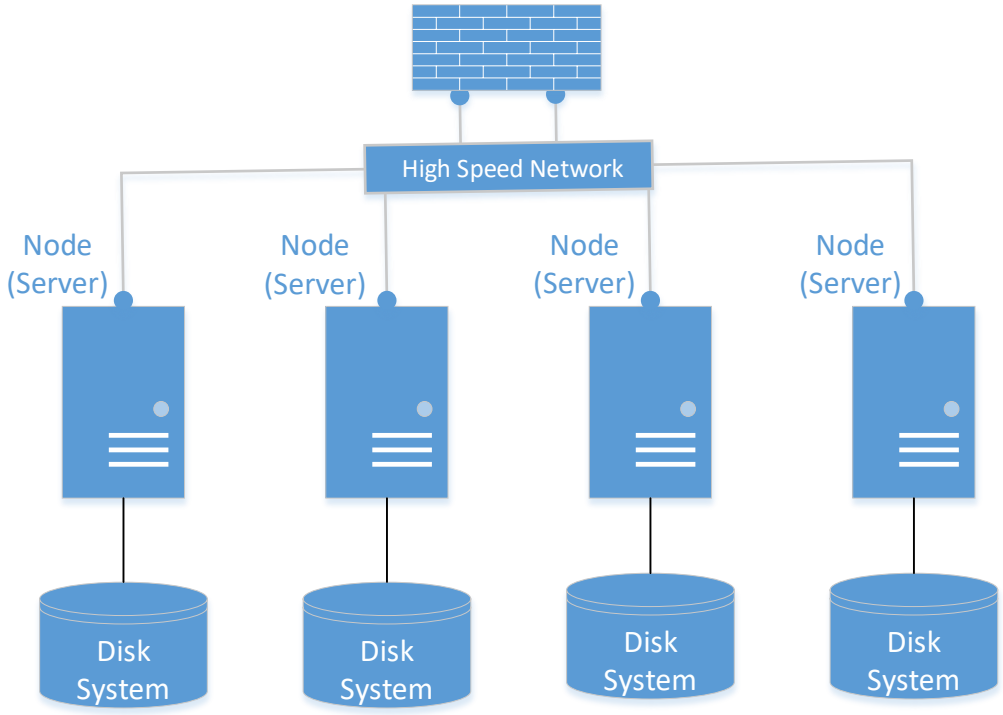
A ***computer cluster*** consists of multiple computers (servers) that are connected (coupled) together in such a way that they can be viewed and managed as a single system. Each server in a cluster is commonly called a ***node***, and in most configurations, each node uses the same hardware and operating system. The network connecting the nodes varies considerably across vendor architectures, ranging from loosely coupled local area network (LAN) connections to tightly coupled high-speed fiber optic networks (so-called "fiber channels"). In a loosely coupled network, the servers must work more independently, with less frequent communication between nodes, while a tightly coupled network allows servers to communicate frequently to coordinate the execution of common tasks.

It is important not to confuse a computer cluster with what some RDBMS vendors call a *clustered index*, which is an index where the table rows are physically kept in sequence according to the indexed column values. While a clustered index might appear on a computer cluster, the two have nothing to do with each other.

While there are many architecture variations in use for cluster computer systems, the three most popular architectures for configuring clusters are: MPP (massively parallel processing), commonly known as the "shared nothing" architecture; SMP (symmetric multiprocessing), commonly known as the "shared everything" architecture; and a hybrid I am calling the "shared disk" architecture.

MPP (Shared Nothing) Architecture

MPP is by far the most popular cluster architecture, largely because clusters can be constructed from commodity hardware. Here is a diagram of a small (4-node) MPP cluster:



The hallmark of MPP is simplicity: nothing is shared across nodes, so each node has its own CPU (central processing unit), memory, and dedicated disk storage system. To make this work for a DBMS, the data must be divided (partitioned) across the nodes, such that each node "owns" a portion of the database. The subsets of the database are called partitions, segments, or shards, depending on the vendor. (I use the term segment to avoid confusion with other uses of the terms partition and shard.) For relational databases, this is typically accomplished using one or more columns in each table, specifying specific values or ranges of values for each node; or alternatively, using a hash function to evenly distribute rows across the available nodes. (A *hash function* is a software module that maps data of arbitrary size to data of a fixed size. Hashes are easier to use for purposes such as indexing or mapping data due to their fixed size.) DBMS products that support MPP clusters include MySQL, Microsoft SQL Server, HP Vertica, Bizgres MPP (based on PostgresSQL), and most NoSQL products.

The disk systems attached to each node can be simple disk drives, or they can be disk storage systems, such as RAID devices, that provide internal fault tolerance such that the failure of a single drive (media failure) does not result in data loss. However, in order to support fault tolerance when an entire node fails, each data segment must be replicated (mirrored) on at least one of the other nodes. DBMSs that support clustering typically provide a facility for defining and managing replicated data to support fault tolerance. An alternative is to allow other nodes to connect to disk systems of failed nodes so that the data is still accessible.

The primary advantages of this architecture are:

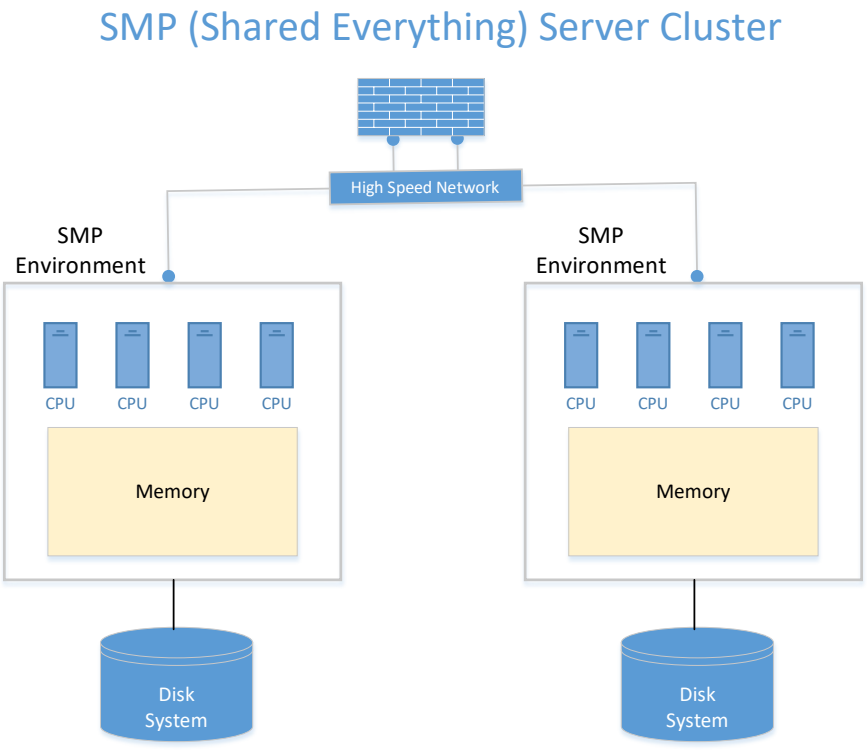
- **Cost**: Hardware/software costs are minimized because commodity (commercial off-the-self) hardware and operating systems are used.
- **Scalability:** The cluster can be expanded easily by simply adding more nodes.
- **Widely Supported:** There are many DBMS vendors that support this architecture.

The main disadvantages of shared-everything clusters include:

- **Cross-node Performance**: If data is not carefully segmented across nodes, operations that require data stored on different nodes must have the data transmitted across the network, which can considerably slow down processing. I present some design guidelines for avoiding this situation later in this topic.
- **Processing Must Be Asymmetric**: For operations that span nodes, such as database queries that require a global view of the segmented data, it is not practical to synchronize query execution across nodes. Therefore, global queries must be split into queries that can be processed independently on each node (with each node querying its part of the data), and the results sent to one of the nodes so that the results can be merged into the final query result set. If data is not distributed evenly, the node that must do the most work becomes the limiting factor.

SMP (Shared Everything) Architecture

SMP (symmetric multiprocessing) systems (commonly known as "shared everything" servers) are, as the name suggests, server systems that contain many CPUs, a common bank of memory shared by all the CPUs, and in its purest form, a common disk system. In contrast, non-SMP servers that contain multiple CPUs, have portions of memory dynamically allocated to each CPU based on need, so memory is not simultaneously shared with all CPUs as it is in SMP environments. The following diagram shows two SMP environments, which would be typical for situations where high availability is required.



This diagram shows the SMP architecture in its purest form, with disk systems dedicated to each SMP environment. Vendors often use the term "environment" instead of "server" because SMP

environments are so much larger and more complex than a typical commodity server. However, the dedicated disk systems make this architecture unsuitable for use with large DBMSs, hence the need for the “shared disk” hybrid, which is presented a bit later in this topic.

The primary advantages of this architecture are:

- **Support for Complex Symmetric Operations**: When performing complex calculations that require a lot of memory and CPU resources, this solution is hard to beat.
- **Higher Performance**: The SMP environments are generally high-powered units that perform well above the levels of comparable MPP systems.

The main disadvantages of shared-nothing clusters include:

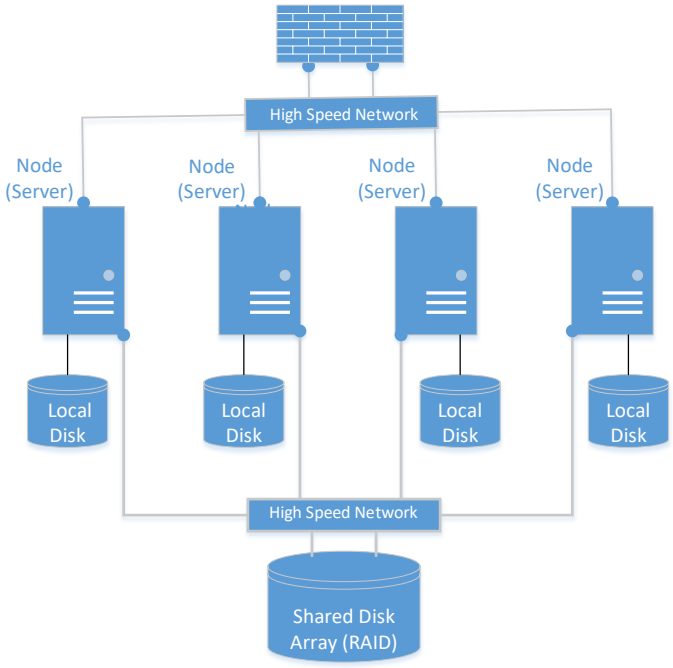
- **Cost**: Hardware/software costs are higher than MPP clusters because specialized hardware is required.
- **Scalability Has Limits:** The servers can be expanded by adding or upgrading internal CPU and memory units, but there is always a maximum amount that can fit within a given environment. In other words, scalability is not as open-ended as with MPP systems.
- **Limited DBMS Support:** The only vendor that supports an SMP database architecture is Oracle. However, Oracle uses the shared disk hybrid of the SMP architecture, which is presented next.

Shared Disk (SMP Hybrid) Architecture

As already mentioned, this architecture uses SMP environments (servers) for each node, and each has a dedicated local disk, but with a shared disk system (typically RAID) added to hold the shared data, including the database data files. The shared disk system is connected to all cluster nodes via a high-speed network. Currently, Oracle is the only DBMS vendor that uses this architecture, which they call Oracle Real Application Clusters (RAC).

Here is a diagram of a Shared Disk cluster:

Shared Disk (Hybrid SMP) 4-Node Cluster



Being a hybrid, this arrangement attempts to optimize the advantages of both the MPP and SMP architectures. In order to minimize contention for data contained on the shared disk system, Oracle RAC maintains a shared cache on each node through a feature they call “Cache Fusion”, which uses the high-speed network to keep the caches in alignment. As with all SMP architectures, the main disadvantage is higher cost.

Physical Design Considerations for Databases Deployed on Cluster Servers

Whenever database data is divided (segmented) across multiple nodes in a cluster, the physical database design should follow some configuration guidelines in order to optimize performance. While these guidelines often vary across particular implementations, here are the basics:

- **Minimize Global Indexes**: In a cluster server environment, a *global index* is an index that contains values for all the rows in a relational table, regardless of the cluster node that contains those rows. Primary key and unique constraint indexes have to be global in order to prevent duplicate values across an entire table. However, you should define local indexes for any non-unique indexes. A *local index*, as the name suggests, is segmented across nodes in the cluster, and thus the index segment that is local to the node contains only the data values for table rows that are physically stored on the local node. Using local indexes not only helps query performance by allowing index searches to be performed in parallel, it also reduces network traffic between nodes when indexed column data is maintained.

- **Avoid Joins Across Nodes**: If you write a join where the rows that must be joined reside on different nodes, data must be transmitted ("broadcast") across the network that connects the nodes to whichever node is handling the task of merging the results into the final result set. Each pair of rows to be joined must be in memory on a single node so the DBMS to assemble the joined row for the result set. Here are the specific guidelines for segmenting tables that are frequently joined:
    - **Joining Large Tables:** When joining two large tables, segment both tables based on the columns being matched by the join predicate (the join key) so that rows being joined are always stored on the same node. For example, if you need to join a one million row Order table and four million row Order Line Item table, you should segment both tables by the join key, which is probably Order ID. By doing so, all the rows for a given order and all of its line items will be physical stored on the same node. This allows the join operation to be run in parallel on each node without the need to broadcast any rows across the network that connects the nodes.
    - **Joining Small and Large Tables:** When joining a small table with a larger one, replicate the small table across all the nodes. As before, this allows all the joins to be done locally. In this situation, the terms "small" and "large" are relative. For example, a table with one or two million credit card account holders would be considered "small" when you consider joining it with a table containing several hundred million rows of credit card transactions.