



A Primer on Database Clustering Architectures

by
Mike Hogan, CEO
ScaleDB Inc.

Overview

Users often implement databases in a clustered configuration in order to accommodate business requirements such as scalability, performance, high-availability and failure recovery. When implementing a cluster, the architecture of the underlying DBMS is an important factor to consider. The two predominant DBMS architectures are shared-nothing and shared-disk. This paper provides a high-level comparison of these two architectures. For a more detailed comparison on the pros and cons of the two, you will want to read the white paper [Shared-Disk vs. Shared-Nothing](#).

Other Complementary White Paper Resources

[Shared-Disk vs. Shared-Nothing](#): An in-depth comparison of these database architectures.

[Cloud Databases](#): Summarizes the database requirements for cloud databases and compares the suitability of different database architectures to cloud computing.

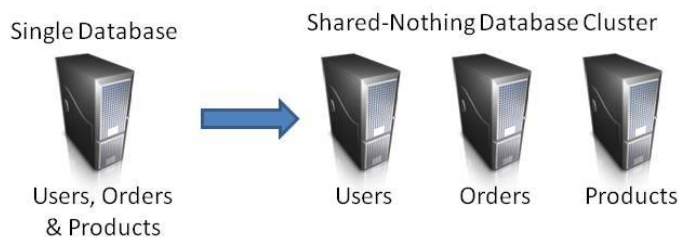
An Introduction to Clustering

Wikipedia describes cluster computing as “a group of linked computers, working together closely so that in many respects they form a single computer.”¹ In the database vernacular, clustering means that the application sees a single database, while under the covers there are two or more computers processing the data. In addition to providing scalability, database clusters can deliver additional benefits such as load balancing and high-availability, but these things are not inherent in all database clusters.

Shared-Nothing

Shared-nothing works on the principle that each node in a cluster has sole ownership of the data on that node. Each node literally shares no data with the other nodes of the cluster, hence the term shared-nothing. When you move from a single server to multiple servers in a shared-nothing cluster, you must divide the data across the servers. This process of splitting the data across servers, as indicated in the diagram below, is called partitioning. Data can be partitioned² vertically or horizontally, but this is outside the scope of this paper.

Diagram 1: Shared-Nothing (partitioning the data across nodes)



Requests for data are then processed through a routing table that routes each request to the server/node that owns that data. For example, the Server 1 above may have information about users, while Server 2 might have information about orders. If your application makes a request that involves both servers, for example requesting a list of users and order information for orders placed in the prior month, you need to involve both servers. The database would solve this request by reading the list of orders placed the prior month and then sending that list from Server 2 to Server 1 to add the information about the users. This process of sending data from one server to

¹ Wikipedia, Computer (cluster): http://en.wikipedia.org/wiki/Computer_cluster

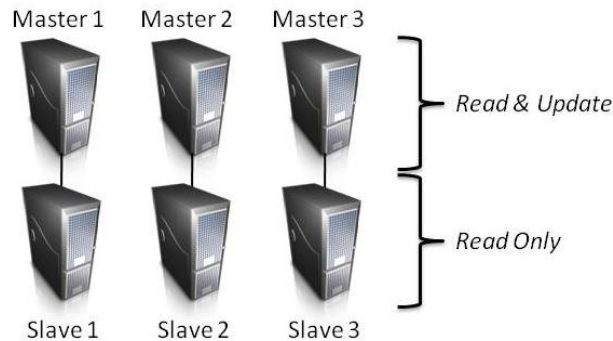
² Wikipedia, Partitioning (database): [http://en.wikipedia.org/wiki/Partition_\(database\)](http://en.wikipedia.org/wiki/Partition_(database))

another is called data shipping. At its core, a shared-nothing database is a standalone database with the added facility for data shipping.

Fail-Over and the Master-Slave Configuration:

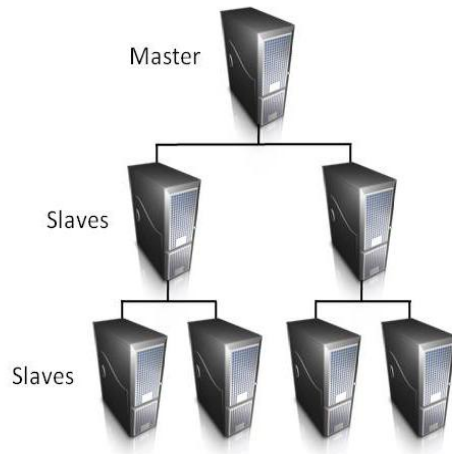
If a server fails, all applications that require access to the failed server also fail. For this reason, each node needs a mirrored copy that can take over in case the primary node fails. This node is called a slave. In the master-slave configuration, the master can perform both reads and updates, while the slave can only provide read access to the slave's local copy of the data.

Diagram 2: Master-Slave Configuration



For applications with a high ratio of reads versus updates, the slaves can be configured in a tree, where the data from the master is replicated down the tree, and the slaves then off-load the read-access.

Diagram 3: Tree of Slaves



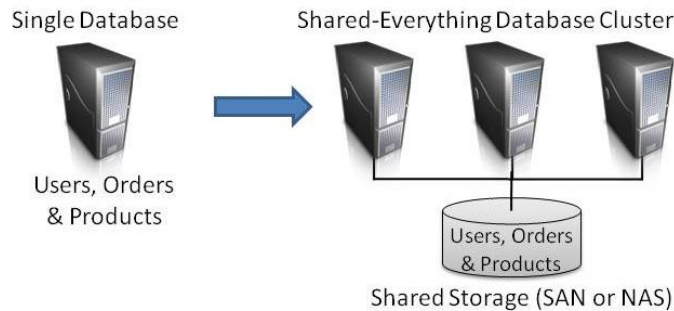
There is nothing to prevent a shared-disk from using a similar tree of read-only slaves, but slaves are generally associated with shared-nothing.

Examples of Shared-Nothing DBMS: Oracle 11g, IBM DB/2 (non-mainframe), Sybase ASE, MySQL (InnoDB, MyISAM, Falcon...all storage engines except ScaleDB), Microsoft SQL Server, etc.

Shared-Disk

Shared-disk, also known as shared-everything, works on the principle that you have one array of disks, typically a Storage Area Network (SAN), or Network Attached Storage (NAS), that holds all of the data in that database. Each server or node in the cluster acts on that single collection of data in real-time.

Diagram 4: Shared Disk Cluster (all nodes have access to all of the data)



In a shared-disk architecture, any node can satisfy any request, because they all have access to all of the data. So, instead of going to a specific node for specific data, shared-disk can simply route the request to the next available node. Because each node can address any database request, the load is inherently balanced across the nodes in the cluster.

Fail-Over, Load Balancing and the Master-Master Configuration:

While shared-nothing has only one node that can update any piece of data in the database, shared-disk enables any node to update the database. For this reason, shared-disk is called a master-master architecture. As such, it fully utilizes each server in the cluster. This architecture also provides inherent failover, since each node acts as a back-up to every other node. Finally, the shared-disk architecture provides inherent load-balancing, since each database request can be sent to any available server.

However, nothing comes for free. In order to enable this flexibility, the nodes communicate with each other to coordinate their activity, specifically: locking, buffering and status. This inter-nodal messaging creates some degree of overhead, but you must balance this overhead against the impact of data/function shipping found in shared-nothing clusters.

Examples of Shared-Disk DBMS: ScaleDB (MySQL), Oracle RAC, Sybase ASE CE, IBM IMS and DB/2 (mainframe). ScaleDB is the only shared-disk solution for MySQL.

Which Architecture is Better?

Unfortunately, such a simple question does not have a simple answer. Each architecture has its pros and cons, which are discussed in more detail in the white paper *Shared-Disk vs. Shared-Nothing*³. In many ways, comparing the two architectures is like comparing automobile transmissions. Shared-disk is analogous to the automatic transmission, because it automates much of the complexity of set-up and maintenance, thus lowering total cost of ownership (TCO). Shared-nothing is analogous to the manual transmission because it provides more granular control in exchange for increased manual effort on the part of the owner. And just as drivers tend to be passionate about their preferred transmission, DBAs tend to be passionate about their preferred database architecture.

³ Shared-Disk vs. Shared-Nothing