GateStor
Data Systems Corporation

EpochPOWER
The GateStor Disruptive Cluster Architecture

Abstract

This paper introduces a new cluster architecture that solves problems associated with currently available cluster architectures and provides additional advanced features.

Introduction

Cluster-based storage systems are becoming increasingly important for both research and industry. These clustered storage systems consist of a networked set of storage systems, with data spread across the units to increase performance and reliability.

Using traditional TCP/IP and Ethernet networks to build these systems is attractive because of their low cost and the ubiquity of such networks. However, building storage systems on TCP/IP and Ethernet poses several challenges. Transferring data over TCP/IP is very inefficient due to the protocol overhead which has a significant impact on performance.

Many storage vendors have adopted a traditional cluster architecture in their storage solutions to improve on single-system limitations. The amount of storage supported and performance and scalability are some of the most common issues that storage vendors are looking to clusters to solve.

However, cluster software packages are very complex with thousands lines of code written specifically to address the problems associated with connecting systems through a network. These problems include: reduced performance, dividing data across multiple systems, handling node failure without losing data, and spilling files over from one node to another when editing or adding data to a node.

Cluster-based storage systems rely on standard TCP/IP-over-Ethernet for client access to data. Unfortunately, when data is striped over multiple networked storage nodes, a client can experience a TCP throughput collapse that results in much lower READ bandwidth than what should be provided by the available network links.

Conceptually, this problem arises because the client simultaneously reads fragments of data blocks from multiple sources that, together, send enough data to overload the switch buffers and adapters on the client’s link. Cluster architectures today are based on connecting nodes through standard networks. This causes reduced performance because systems can transfer data at rates that exceed what storage networks using this approach can handle.

Background

In cluster-based storage systems, data is stored across the entire cluster to improve reliability, total storage capacity and performance. Typically, their networks are characterized by high bandwidth (1-10 Gbps) and low latency (round-trip times of 10s to 100s of μseconds) with clients separated from storage servers by one or more switches.

In this environment, data blocks are striped over a number of storage systems, such that each system stores a fragment of a data block, denoted as a Server Request Unit (SRU) and shown in Figure 1.

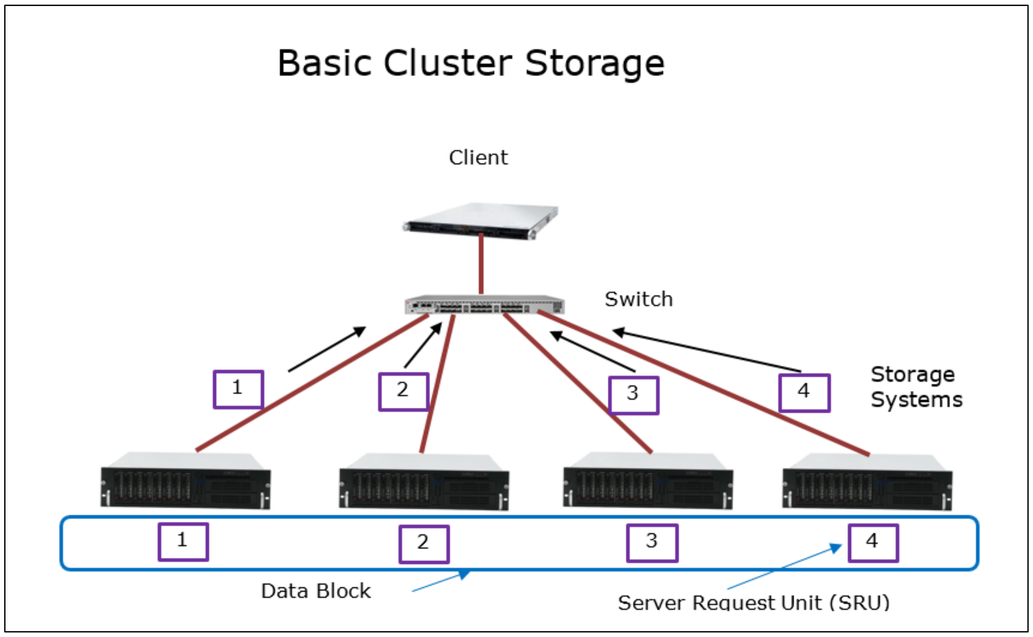

Figure 1

A client requesting a data block sends request packets to all of the storage systems containing data for that particular block. The client requests the next block only after it has received all the data for the current block. We refer to such READs as synchronous READs.

This simple environment abstracts away many details of real storage systems, such as multiple stripes per data block, multiple outstanding block requests from a client, and multiple clients on a single switch making requests across a shared subset of storage systems. However, this is the most basic representation of storage cluster system architecture.

Most networks are provisioned such that the client's bandwidth to the switch should be the throughput bottleneck of any parallel data transfer. Unfortunately, when performing synchronized READs for data blocks across an increasing number of servers, a client may observe a TCP throughput drop of one or two orders of magnitude below its link capacity.

Figure 2 illustrates this performance drop in a standard cluster-based storage network environment when a client requests data from just seven servers. This problem is known as Incast and is attributed to multiple senders overwhelming a fixed-size switch buffer.

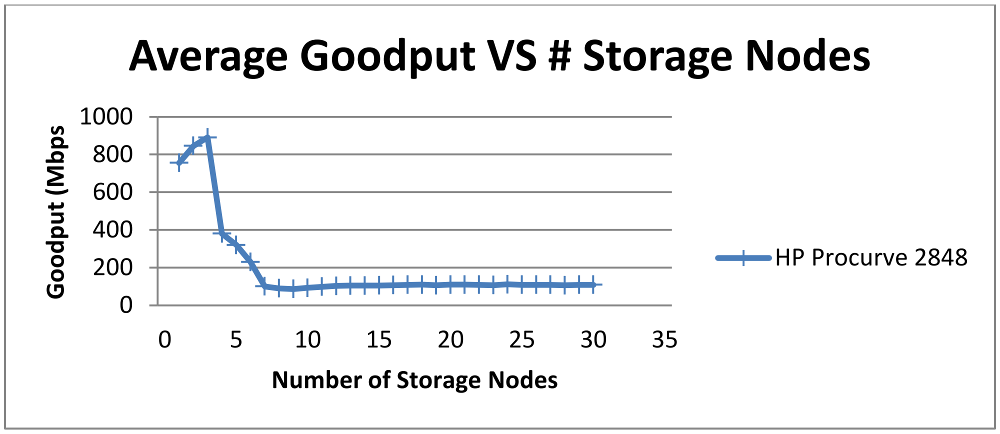

Figure 2

Incast prevents packet losses by creating timeouts, slowing data gathering. Incast can be avoided by increasing the buffer space allocated per port on switches. In fact, larger buffer size on switches delays the onset of Incast.

Figure 3 shows that doubling the size of the switch's output port buffer doubles the number of servers that can transmit before the system experiences Incast. With a large enough buffer space, Incast can be avoided for a certain number of storage systems, as shown by the 1024KB buffer line in Figure 3.

This is corroborated by the fact that we were unable to observe Incast with 87 servers on the Force10 E1200 switch, which has very large buffers. But Figure 3 shows that for a 1024KB buffer, 64 servers only utilize about 65 percent of the client's link bandwidth—and doubling the number of storage systems only improves application-level throughput (Goodput) to 800Mbps using a Gigabit Ethernet network.

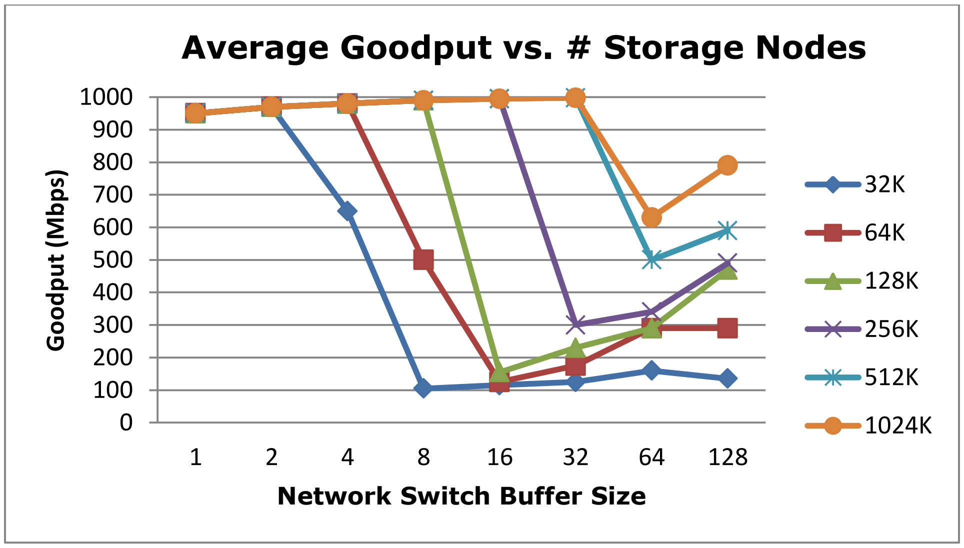

Figure 3

Unfortunately, switches with larger buffers tend to cost more (the E1200 switch costs over $500,000 USD), forcing system designers to choose between over-provisioning, future scalability, and hardware budgets. This suggests that a more cost-effective solution is needed to address the problem of Incast-caused timeouts.

Dynamically limiting the storage cluster to a defined number of active nodes has been shown to be the best approach to avoid Incast, but it is very difficult to implement in storage clusters.

The need for a high performance environment that supports parallel operations such as synchronized READS is particularly important when files instead of blocks are segmented. Recent proposals as parallel NFS (pNFS), a component of NFSv4.1, can support parallel data transfers.

There are basically two file-level cluster deployments. One method segments the global volume on smaller virtual volumes contained on each node. The second uses a file-system-like pNFS to stripe data across file systems in a traditional storage cluster environment.

When using file systems to store data across the cluster, many problems become evident. Some of these problems are dependent on the file size and can dramatically affect the performance of the cluster. Both file cluster implementations generate serious issues. Striping data across nodes can cause Incast timeouts. Segmenting the volume has file-size limitations and over-spillage. Both of these limitations are overcome by the Epoch cluster architecture.

When you distribute a data set by fragmenting it across individual nodes in the cluster, each node owns its own storage. Metadata directories are then generated indicating where the data is allocated throughout the cluster. These metadata directories often require redirection of the client request to the node that holds the requested data, increasing the network usage and diminishing total throughput.

One of the most complex problems occurs when data in one of the nodes is added and suddenly data spills over to the adjacent node. This is not easy to solve. The solution requires thousands of lines of code, many data transfer operations, and it introduces a huge performance hit when it happens.

Also, this common cluster architecture creates the problem that data needs to be replicated to other nodes in the event that a particular storage node becomes inaccessible. The amount of storage used for this replication is dependent on the redundant cluster implementation, but it can be significant and very expensive.

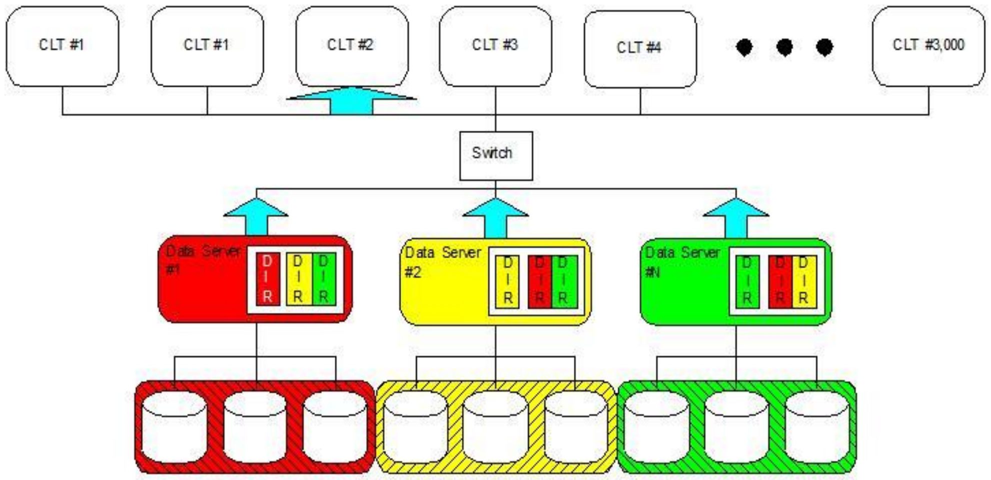

Figure 4

Depending on the file cluster implementation, the architecture may not play well with the application file size. Some clusters are not designed to handle large files that do not fit on a single node, thus creating problems for applications—such as petroleum exploration or weather analysis and forecasting—that generate huge files.

The Solution

In this paper we are introducing a completely new architecture for clusters that avoids many of the current issues and significantly reduces the complexity of both small and large clusters. This innovative approach solves the problems we have discussed and does not impose the limitations that network clusters typically have.

This ground-breaking approach, shown in Figure 5, connects the cluster nodes via the PCI Express (PCIe) bus—not through the standard network. By connecting the PCIe across all the nodes, each component connected to the unified bus has access to any of the devices connected to the PCIe. This not only allows node-to-node data transfers that are an order of magnitude faster, but it also alleviates many issues in accessing previously non-shared resources that now can be controlled by any of the interconnected nodes. Connections to existing legacy networks are still maintained for clients to access the storage network locally or remotely.

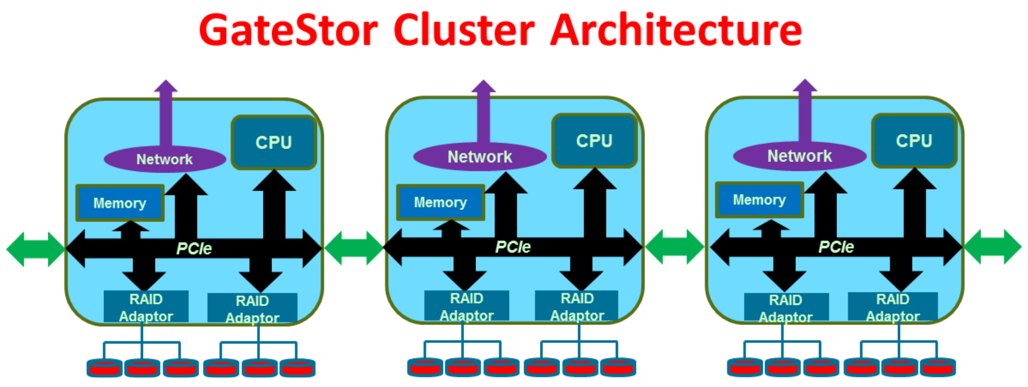

Figure 5

What is revolutionary is the method by which multiple computers or nodes are interconnected to form a cluster. Traditionally, clusters are created by interconnecting nodes via communication networks. The overhead on network protocols affects performance severely and poses many problems that can only be solved by thousands of lines of code—resulting in very complex and sizeable cluster software packages.

In order to make a unified bus, GateStor has developed a PCIe Host Bus Adapter (HBA) that extends the PCIe bus (see Figure 6). This HBA can be inserted in a standard PCIe slot and interconnect the PCIe bus with all the cluster nodes.

In doing this, all the storage data servers can share all the resources connected to any PCIe slot on the cluster. For example, data server node 1 can access the RAID controllers that are in data server node 3, so this technology breaks the physical limitations of an isolated data server and opens up transparent communications with all the nodes behaving in a manner very similar to a parallel supercomputer system.

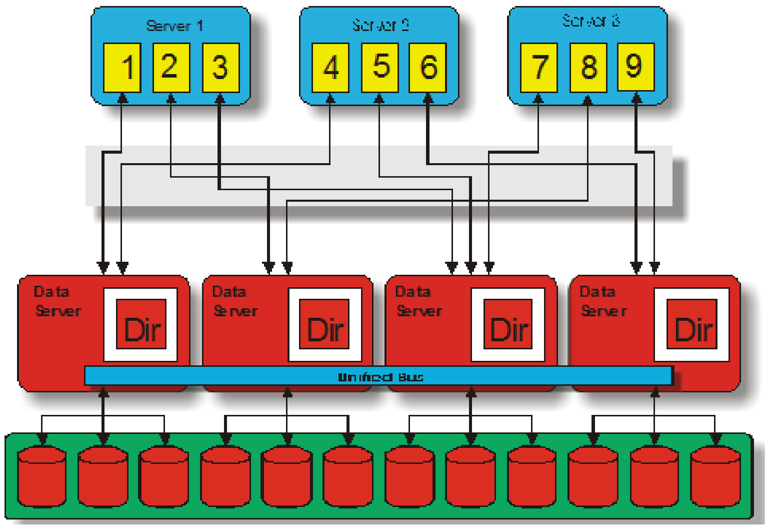

Figure 6

GateStor's patented cluster architecture avoids all the issues we have discussed. The GateStor cluster is not established at the network level, rather at the server or computer bus level, avoiding the need for network protocols and providing many advantages in the cluster environment. Let's look at how this disruptive new architecture addresses the problems that we previously discussed:

- **Reduced performance** due to network protocols and overheads. GateStor avoids this by connecting all nodes to the PCIe instead of a network, thus avoiding any network protocols such as TCP/IP. The bus is the lowest level of I/O interconnection avoiding all networking overhead.

- **Segmenting data** to each cluster node. By using a unified bus, GateStor can ensure that all the nodes in the cluster can share any resource connected to the cluster. The processor of any node can see all the collective memory and adapters from any node connected to the unified bus, avoiding the need for segmenting data. Any node can access the storage of any other node. Even if it is not originally assigned to that particular node, it becomes a common storage pool.

- **Node-to-node data spillage**. Because there is no need to segment data to specific storage nodes, data can grow across the common storage pool and any node can have access to it without the limitation that it had to fit on the storage of a single node.

- **Duplicating data** to different nodes to avoid data loss in case of node failure. With the GateStor solution, data duplication is not required because data can be accessed by any node. With earlier solutions, nodes had dedicated storage, and if the node failed, the storage became unavailable. Thus having two or more copies of the data was a requirement. In our case it is an option and only serves to increment performance. A second benefit of not needing to duplicate is the cost savings. Avoiding duplication of data will save dramatically on cost of storage and power.

- **Improved availability and load balancing**. Data availability with the GateStor solution is vastly improved, because any node can access to any storage server, so it can handle multiple points of failure and still continue to operate correctly. Load balancing can be handled more easily because any node can take over any host request or connection and can be changed dynamically without requiring prior setup.

- **Incast timeouts**. Incast depends on the number of synchronous requests and how the switch buffer is able to handle the data. Incast can be mostly avoided using the GateStor architecture. Dynamic sub-clusters can be created to serve a specific number of host systems. It is important to point out that even if you dynamically define a smaller sub-cluster within a cluster, that sub-cluster can access all the data storage available to the larger cluster. This feature would be very difficult to implement in legacy systems, because they do not support dynamic cluster configuration.

- **Performance limitations**. Networks are still slow and are usually the source of any cluster bottlenecks. With GateStor's approach of interconnecting the data servers and the host servers via the PCIe, throughput will be at least 5 times better than the fastest network communication transfers currently available.

In addition to all the features mentioned above, PCIe broadcast transfers (a patented GateStor technology) bring a unique level of sophistication, innovation and extensibility. The patented technology allows moving data in a single transfer, not only within a storage node, but across all nodes connected to the unified PCIe bus. This avoids double and triple transfer operations that occur in any storage system (to memory, and from memory to the RAID controllers).

In a cluster, this technology becomes more relevant, because the data can be accessed by any node—avoiding many duplicate transfers and allowing multiple copies of data without having the overhead of copying the data from one node to another. This process on other systems could take hours or even days.

Figure 7 shows how data is moved from an I/O interface to the collective memory of all the nodes in the cluster.

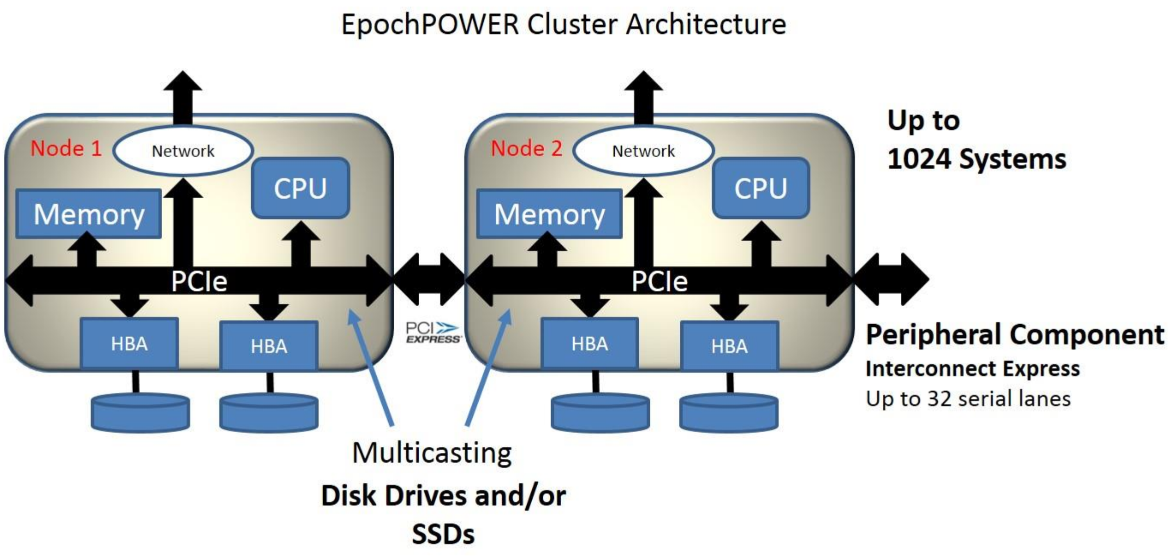

Figure 7

The GateStor PCIe Unified Bus HBA can also be used on the client servers in order to completely avoid the network and all the performance issues that are inherent in network transfers (Figure 8). By connecting the host or client computers to the unified bus, they can share resources not only with the host servers, but also with the storage servers, providing many benefits, especially when using virtualization solutions, which could dramatically benefit from the GateStor technology.

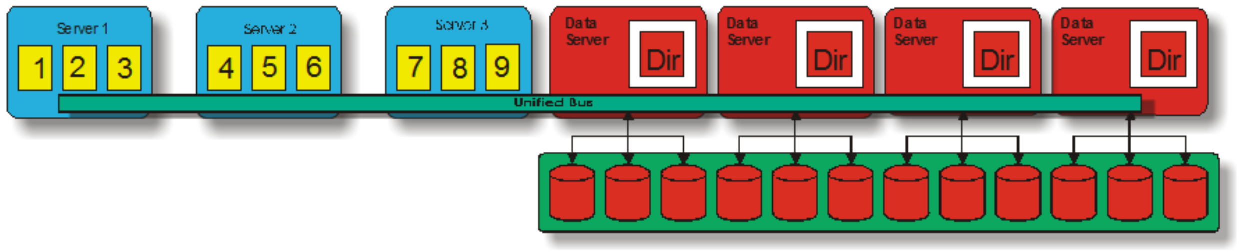

Figure 8

The GateStor PCIe Unified Bus HBA has a dual port to either connect a redundant port to a single node or to connect to two different nodes. Multiple HBAs can be used on a single node to create more connectivity and/or to improve performance.

Directly connecting the PCIe point to point with a cable works well with small clusters, but GateStor has developed a 32 port PCIe Gen-3 switch that can be used for node and host server connection which can be stacked to accommodate a cluster of up to 1024 nodes. Also, multiple switches can be used as a redundant path to all clusters or servers.

Storage Cluster and Servers Connected Through the PCIe Bus

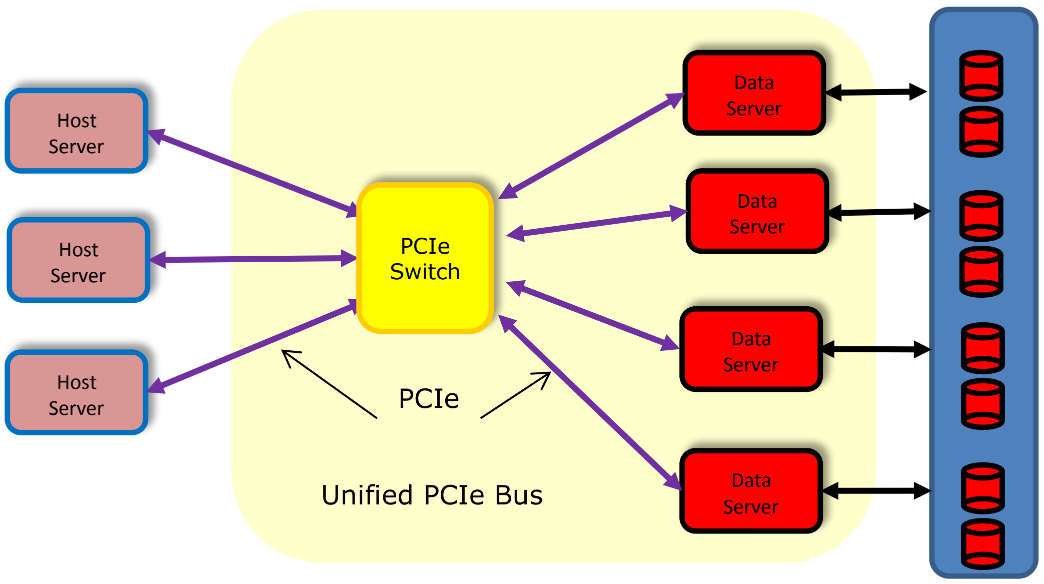

Figure 9

Conclusion

This new cluster architecture brings an innovative approach for interconnecting computers that should become a standard. We also believe that server motherboards will incorporate PCIe connectors and extenders in order to enable interconnection to other mother boards and PCIe peripherals. Companies like Intel are recognizing that PCIe is the most efficient method of interconnecting external peripherals today. And we're not just looking at HBAs. We now see PCIe SSDs which are being offered by several companies.

Furthermore, if GateStor's broadcasting techniques are applied, significant improvements in performance for general data clusters can also be achieved. It will also be possible to build clusters that can handle data and storage selectively and dynamically change the configuration to dedicate active nodes for data crunching or for storage applications.