Map/Reduce on Lustre

Hadoop Performance in HPC Environments

Nathan Rutman
Senior Architect, Networked Storage Solutions

xyratex

Notices

This work was carried out by Xyratex under contract with the U.S. Naval Research Laboratory.

For more information please contact marketing@xyratex.com

Version 1.4 | 2011

Abstract

Map/Reduce is a distributed computational algorithm, originally designed by Google, which is growing in popularity and use for a wide variety of large-scale jobs. The most common implementation of Map/Reduce is as a part of the open-source Hadoop system. By default, Hadoop uses the Hadoop Distributed File System (HDFS) as the underlying storage backend, but it was designed to work on other file systems as well. Various attempts have been made to use Hadoop on distributed file systems such as Lustre, PVFS, Ceph, and GPFS, but a careful study of the architectural tradeoffs among these file systems has not been pursued, and comparative Hadoop performance on clustered file systems has been disappointing.

A fundamental assumption of the Hadoop system is that, "moving computation is cheaper than moving data" [1]. This means that it is more efficient to perform computations on nodes that locally store the data involved rather than move the data to an arbitrary set of compute clients over a network. This premise is a major architectural driver of HDFS, and it has resulted in very good Hadoop performance on "commodity" (i.e. inexpensive) clusters with relatively slow network fabrics (e.g. 1 GigE).

However, in the environment of high-performance computing (HPC), where system performance is of greater (often paramount) importance, the "moving computation" assumption no longer holds true. In many cases, using a distributed file system to leverage the faster networking infrastructure typically available in an HPC environment enables higher overall Hadoop performance than "moving the computation".

In this paper, we compare HDFS and Lustre architectural drivers and resulting system performance of Map/Reduce computations on HPC hardware. We evaluate theoretical and actual performance of Lustre and HDFS for a variety of workloads in both traditional and Map/Reduce-based applications. Further, we examine the additional benefits (cluster efficiency, flexibility, and cost) of using a general-purpose distributed file system, such as Lustre, on a Hadoop compute cluster.

Background

Map/Reduce

Map/Reduce is a framework to easily write applications that process large amounts of data in parallel on clusters of compute nodes. Generally, in a Map/Reduce environment, the compute and storage nodes are the same, that is, computational tasks run on the same set of nodes that permanently hold the data required for the computations.

The Map/Reduce algorithm breaks input data into a number of limited-size chunks ("splits"), on a parallel basis. The algorithm converts the data in each split to a group of intermediate key-value pairs in a set of Map tasks. Then, it collects each key's values (a key may have multiple values), in what is called the "shuffle" stage and processes the combined key/values as output data, via a set of Reduce tasks.

In traditional Map/Reduce environments, input and output data are stored on the HDFS, with intermediate data stored in a local, temporary file system on the Mapper nodes, and shuffled as needed (via HTTP) to the nodes running the Reducer tasks.

1 The Apache Software Foundation. Apache Hadoop Project, http://hadoop.apache.org/common/docs/r0.20.2/hdfs_design.html

The Map/Reduce framework is written in Java, making it easy to deploy across operating systems and (commodity) hardware platforms, and it implements a library of functions and default values at each processing stage.

HDFS is not a POSIX-compliant file system, and once data is written it is not modifiable (a write-once, read-many access model). HDFS protects data by replicating disk data blocks across multiple devices. By default, HDFS data blocks are replicated three times: (local, "nearby", and "far away") to minimize the impact of a data center catastrophe. A new network striping module for HDFS, created by Facebook, is now available.

Lustre

Lustre is a client/server based cluster file system. Data is stored on Object Storage Servers (OSSs) and metadata is stored on Metadata Servers (MDSs). Lustre is designed for large-scale compute and I/O-intensive, performance-sensitive applications. It is optimized to operate efficiently on many types of high-end network fabrics, including InfiniBand, Elan, Myrinet, as well as TCP/IP, taking advantage of RDMA where available. Files in Lustre are broken into stripes, which are typically stored on multiple Object Storage Targets (OSTs), allowing parallel read and write access to different parts of the file. Lustre is POSIX-compliant and mounted remotely in a manner similar to NFS.

Previous Work

The use of clustered file systems as a backend for Hadoop storage has been studied previously. The performance of distributed file systems such as Lustre[2] , Ceph[3] , PVFS[4] , and GPFS[5] with Hadoop has been compared to that of HDFS. Most of these investigations have shown that non-HDFS file systems perform more poorly than HDFS, although with various optimizations and tuning efforts, a clustered file system can reach parity with HDFS. However, a consistent limitation in the studies of HDFS and non-HDFS performance with Hadoop is that they used the network infrastructure to which Hadoop is limited, TCP/IP, typically over 1 GigE. In HPC environments, where much faster network interconnects are available, significantly better clustered file system performance with Hadoop is possible.

HDFS Design Choices

Design for Map/Reduce

HDFS was written specifically for Map/Reduce compute jobs, with the following design factors in mind:

- Use a write-once, read-many (WORM) access model. Generally, in Map/Reduce processing jobs, input data is static and output data is separable – the output of each Reduce task depends only on a limited subset of the keys. These jobs are usually not iterative, so they tend not to have random-access write patterns that continuously modify a common solution space.
- Use commodity hardware with the expectation that failures will occur. Hardware (compute nodes and hard drives) is assumed to fail on a regular basis. Rather than isolate this issue as a hardware problem to be mitigated

2 Sun Microsystems, Inc., "Using Lustre with Apache Hadoop", 2010.

3 C. Maltzahn, et al., "Ceph as a Scalable Alternative to the Hadoop Distributed File System", ;login:, 35(4):38-49, 2010.

4 W. Tantisiriroj, et al., "Data-intensive File Systems for Internet Services: A Rose by any Other Name...", Carnegie Mellon University Parallel Data Lab, Technical Report CMU-PDL-08-114, 2008.

5 R. Ananthanarayanan, et al., "Cloud Analytics: Do We Really Need to Reinvent the Storage Stack?". In HotCloud '09.

with “expensive” solutions (e.g. RAID systems, non-interruptible power, failover servers, etc.), HDFS uses a timeout strategy to reassign tasks after a hardware failure. If an unfinished Map or Reduce task exceeds the specified timeout, then its node is assumed to have had a hardware failure, and the task is restarted on a different node. To maintain data integrity in an environment with hardware failures, HDFS replicates data blocks in multiple locations.

- Access data in long, sequential streaming data reads. The Map/Reduce framework is designed for job types requiring data to be read in large, contiguous data blocks rather than random-access patterns.
- Process very large files. HDFS assumes that input data is collected in large chunks.
- Use a file system that is hardware and OS agnostic. To support the largest variety of installations with a minimal number of dependencies, HDFS code is written in Java as user-space code.
- Moving computation is cheaper than moving data. The HDFS documentation specifies, “A computation requested by an application is much more efficient if it is executed near the data it operates on”. HDFS tries to schedule jobs on nodes that have local access to the on-disk data.

Implications for Performance

While these design criteria are appropriate to process Map/Reduce jobs in many situations, they can significantly impair Map/Reduce performance in I/O-sensitive cases, particularly when HPC compute environments are used:

- Uncached data. When large, contiguous streaming data is delivered, the effect of caching data is inconsequential. But when the read load is random, seek-intensive, or partial-block, the lack of data caching is, potentially, a high cost performance deficit.
- Slow file modifications. In a WORM model, changing a small part of a file requires that all file data be copied, resulting in very time-intensive file modifications (other than simple appends).
- Using HTTP instead of a high-speed network fabric. To support a wide variety of environments and hardware, network data transfer in Hadoop uses standard HTTP for the shuffle stage. Employing HTTP prevents Hadoop from taking advantage of Direct IO, Direct Memory Access (DMA) or the latency improvements available in faster network fabrics.
- Data replication costs time. In Hadoop, every data block must be written remotely, to ensure that all blocks are duplicated, and it must be written “far” across the network (ideally to another data center) for ultimate safety. Moreover, these over-network writes are synchronous; the client writes blocks until all copies are complete, on disk. Therefore, write time is gated by the longest latency path - two network hops plus a disk write for 3x replication. While this write method would be disastrous for a general-purpose file system, with true WORM usage it may be acceptable to penalize writes this way.
- Moving computation is cheaper than moving data. This statement can be made more accurate by comparing local data movement times and the time for data to traverse a network. The outcome depends primarily on storage and network bandwidth availability.

Implications for Scalability

Although scalability is a primary goal of the HDFS design, several design factors impose scalability limits, complicating this objective.

- Large block size implies fewer files. HDFS reaches system limits in the tens of millions of files.
- Data replication costs bandwidth. In HDFS, every data block is sent over the network twice, by default. On this basis alone, compared to a distributed, clustered file system like Lustre, HDFS can optimally achieve only half the write performance in a network-limited environment. (Replication and backup strategies for Lustre are typically asynchronous.)
- Purely from an I/O perspective, a balance between network and storage bandwidth, together with CPU capability to exploit it, avoids idle subsystems. In turn, idle subsystems imply a lost opportunity cost.
- Data replication costs storage. Server-based clustered file systems, like Lustre, are designed to modularize failure domains in hardware to achieve data safety and availability. Battery-backed RAID systems, failover server pairs, and redundant networking provide safeguards against component failure. Typically, in these systems, excess disk capacity is designed for ~ 20% RAID disk overhead for parity and hot spares. By contrast, default 3x HDFS data replication costs 200% overhead. (Note that the recent addition of HDFS RAID can change the replication cost for HDFS. When enabled, parity data is calculated and stored for each file, allowing reduced replication factors while maintaining data safety; in practice, a 2.2x replication factor yields an effective factor of 3.x. This reduced replication factor comes at a price, in terms of CPU and network usage for parity file calculations and storage.) While data safety is qualitatively different between data triply replicated in distinct data centers, versus RAID-backed in a single unit, this must be weighed against the additional storage cost required by the former. For large scale systems, it may be more cost-effective to ensure data safety with dedicated, modularized hardware than with drives.

Implications for Storage

Finally, several attributes of the HDFS design are incompatible with the storage environments of HPC clusters.

- HDFS is not POSIX-compliant and, therefore, behavior outside of Hadoop is not clearly predictable.
- HDFS is not a general-purpose file system. HDFS disks cannot be used by non-HDFS-aware applications – most notably, large simulation applications typically used in HPC environments. Using a common file system for Map/Reduce and other HPC applications potentially saves duplication and segmentation of storage capacity.
- Data transfer is required for secondary analysis. Visualization of Hadoop results or secondary, non-Hadoop analysis of results requires data transfer out of HDFS and into a file system (typically, a POSIX-compliant networked file system), where secondary applications can interact with the data. Similarly, before Hadoop jobs can run, the input data must be transferred into HDFS (or a specialized input module may be written). These required transfers result in potential duplication of data as well as a time penalty for the additional data swaps.
- Data replication costs storage, as described above.

The Case for Moving Data Over the Network

In the everyday user experience, reading data from a local hard drive on a PC is much faster than reading data over a network (typically, a 1 GigE link today). But with the high-speed network fabrics used in today's HPC clusters, data can be delivered over a network at much higher rates than a commodity local hard drive.

The following simple example illustrates this point. We assume a cluster of 100 compute nodes and 100 - 1TB hard drives with a typical usable I/O rate of 80MB/s. The cluster is connected with a modest 4xSDR InfiniBand fabric at 8Gb/s or 1GB/s throughput point-to-point. In Figure 1, the line weights are proportional to the bandwidth capacity.

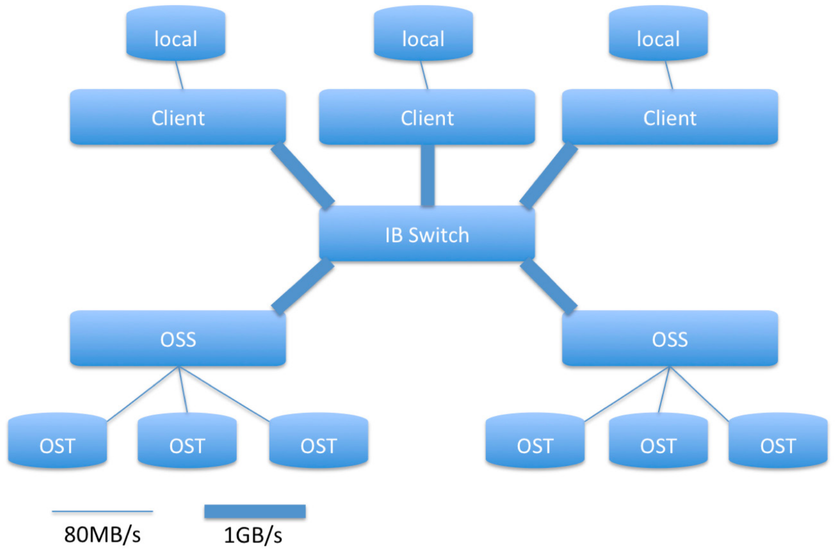


Figure 1: Example HDFS/Lustre cluster layout

- If the cluster is arranged with a Hadoop/HDFS topology, one drive is placed in each compute node, for 100 * 80MB/s = 8GB/s combined maximum throughput to disk. (The drives are marked "local" in Figure 1.)
- If the cluster is organized for a Hadoop/Lustre environment, the drives might be arranged as 10 OSSs, each with 10 OSTs of one drive each. Each OSS can access its 10 OST drives simultaneously, enabling the OSS to read 800MB/s from the drives, for the same total combined 8GB/s throughput to disk. But each OSS should be able to deliver 1000MB/s over the network in parallel. The limitation on bandwidth comes from the drives, not the network – system performance when moving data over the network should be identical to reading data from local drives.

Although a different arrangement of OSSs and network fabric might result in a different throughput limit, the fundamental point remains true – with careful selection of network capacities, there is no absolute advantage to using local drives.

In fact, the illustrated example is a worst-case scenario for network requirements, by allowing sustained maximum throughput to all clients simultaneously. In actual Map/Reduce scenarios, Mapping tasks finish at different times (and Reduce tasks ultimately block, waiting until all of their data is available). Further, most Map tasks require significant compute time and are not continuously executing I/O operations.

Where compute time is a larger percentage of the overall task time, the total task time can be significantly improved by "aggregating" the idle disk bandwidth via striping. For example, assume that each OSS contains data for a set of

10 dedicated compute node clients. If each Map task in the group of 10 compute nodes creates output staggered over time, then each node can use the aggregated disk bandwidth of 800MB/s, by striping the file across all 10 OSTs on the OSS. This write rate is 10 times faster than in the single-drive-per-node case. Thus, for example, a task that uses local disk and consists of 20 seconds input, 60 seconds compute, and 20 seconds output should become 20 seconds input (to the last client), 60 seconds compute, and 2 seconds output, as long as the 2 seconds output does not overlap with one of the other 9 nodes. (Input data will typically not need to be "manually" staggered, because compute time will vary naturally from node to node.) Figure 2 shows the effect of aggregating disk bandwidth.

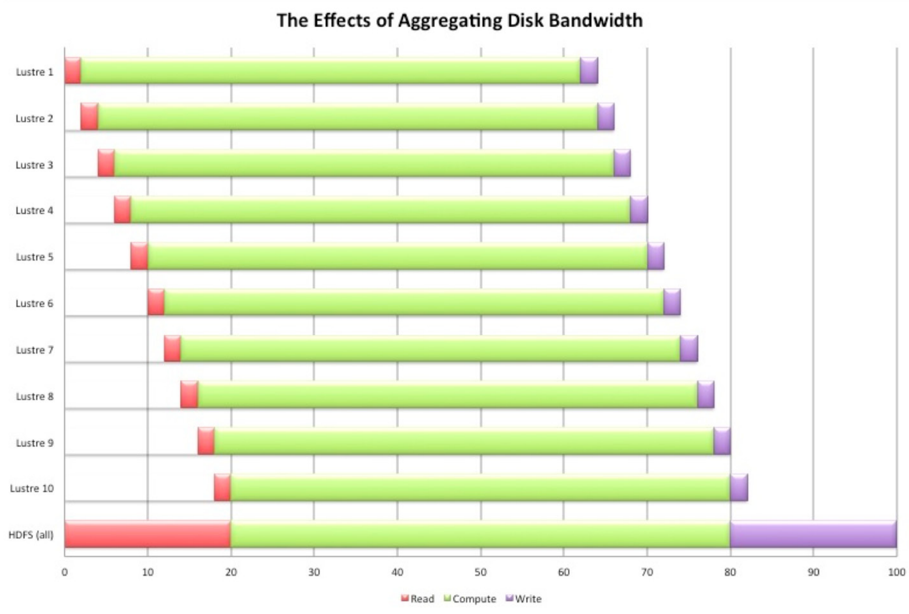


Figure 2: Theoretical effect of aggregating disk bandwidth

To conclude, network limits should not restrict data access speed and, by harnessing the powerful ability to stripe files across drives, we can deliver network-based I/O performance that is significantly higher than local-disk performance.

Test Results

TestDFSIO

To validate the propositions presented in this paper, we used TestDFSIO, an I/O-intensive Hadoop standard benchmark, to test the backend file system performance of HDFS and Lustre in the context of a Hadoop job. For the Lustre test runs, we introduced a small patch in Hadoop to replace the HTTP transport in the shuffle phase with a simple hardlink to the Map task output data. No other changes were introduced into Lustre or Hadoop. We believe significant opportunity still exists to further optimize the Hadoop codebase for clustered file system performance, but we wanted to avoid any significant modifications of the code for this comparative study.

For the test environment, we used a small cluster of 8 nodes with both an overprovisioned network (4xQDR InfiniBand), and a 1 GigE network to investigate the effect of having an unrestrictive network bandwidth limit. For the Lustre setup, we used 2 OSSs with 4 OSTs of a single drive each. For the HDFS setup, we installed one of the same drive types in each of the 8 client compute nodes. All drives measured ~80MB/s for sustained I/O throughput.

For these tests, we used the default settings for Lustre and Hadoop. For Lustre (v. 1.8.0), all files are fully striped. For the Hadoop (v. 0.20.0) TestDFSIO job, we use Max Reduce Thread = 1; File Size = 512MB.

As described previously, this test was expected to be a worst-case scenario for the relative performance of Lustre, as it effectively runs I/O to all clients full-time. The TestDFSIO performance results are shown in Figure 3.

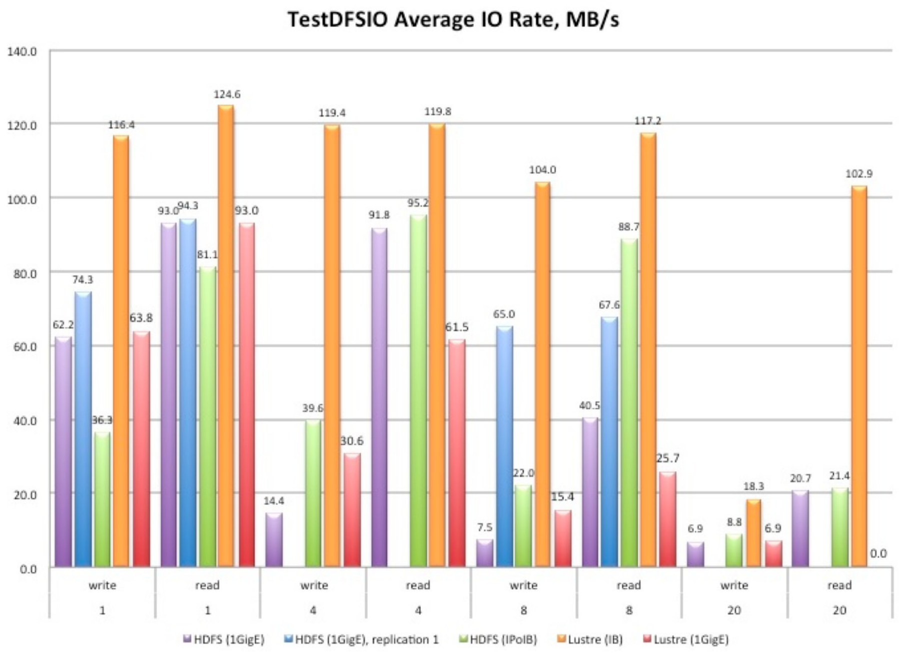


Figure 3: TestDFSIO performance on Lustre vs. HDFS

The TestDFSIO benchmark was run under various loading conditions, from single-client usage (1 Map task) to full usage (8 Map tasks), as well as an overloading case (20 Map tasks). The test results were in line with our predictions: Lustre significantly outperformed HDFS when using a fast network. Under the full-usage load conditions, when all 8 clients are used simultaneously, the difference between HDFS and Lustre performance is the most dramatic. The HDFS write rate for 3x replication drops to 7.5MB/s on 1 GigE and 22MB/s for IPoIB, most likely due to extreme network load for transferring two copies of all data over the network. The write rate for 1x replication is significantly faster (65MB/s), slightly under the disk limit of 80MB/s. Because of striping, the Lustre results are not limited by single drive speed, and recorded a write rate of 104MB/s over the IB network.

Note that in all cases, when using Lustre with the 1 GigE link, Lustre performance is more or less even with HDFS, as seen in previous studies. Lustre's average performance on 1 GigE is the result of the shared bandwidth architecture of Ethernet (no switch was used in this testing), and the overall limited speed of 1 GigE. With 8 nodes reading data at 80MB/s from local drives, the HDFS total bandwidth is 640MB/s, compared to the 100MB/s total limit imposed by the network for Lustre.

For more lightly-loaded runs, the results are less dramatic, but Lustre still shows a clear advantage on a single client, again likely because of the striping benefit. Tests were run with 20 Mapper tasks processed on 8 nodes to determine the effect of significant drive contention. Read caching on the Lustre clients most likely accounts for the continued good performance of read operations under these conditions.

Sort

We ran the Hadoop standard sort benchmark on an identical set of 3.2GB RandomWriter data on HDFS and Lustre, 1 GigE and IB. This test has a higher compute component than the TestDFSIO benchmark. The results of this test, shown in Figure 4, approximate the outcome of the heavily-loaded TestDFSIO case; Lustre throughput on InfiniBand is about 3 times faster than HDFS and Lustre 1 GigE.

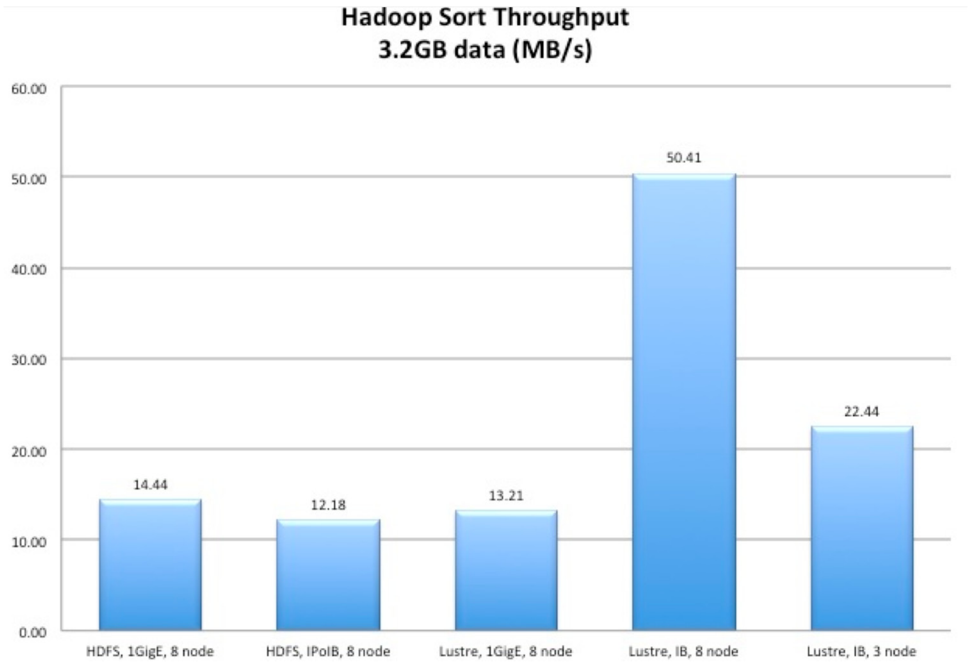


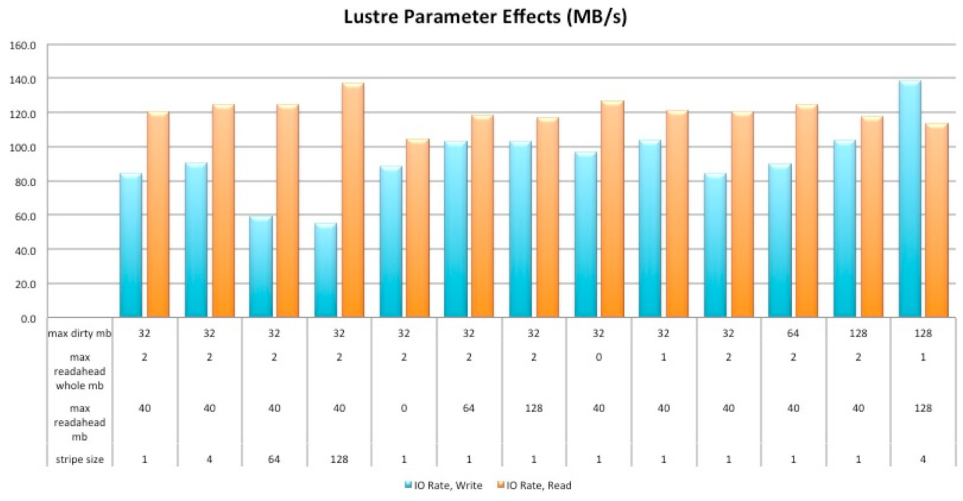
Figure 4: Hadoop sort throughput of 3.2GB random data

Finally, we ran the same sort on 3 nodes of our 8-node cluster (roughly 1/3 of the nodes) to test the proposition that a higher-performance cluster can be used in place of a larger node count. In our test, Lustre on IB on 3 nodes is about 55% faster than HDFS on 8 nodes (1 GigE and IPoIB).

The sort benchmark represents one typical use case, but clearly different usage patterns will be affected to varying degrees. Also, the performance of a Hadoop job can be highly sensitive to Hadoop parameter tuning, see e.g. the Berkeley Performance Tuning Study (we ran all of our tests untuned). Still, the performance of Lustre over IB is remarkable; in effect, this implies that a Lustre/IB cluster could perform the same Map/Reduce job in the same time with one third as many compute nodes.

Lustre Tuning Effects

We also measured the effects of tuning various Lustre parameters for improved Hadoop performance. HDFS uses 128MB block sizes and tuning Lustre parameters to match those expectations yields some beneficial results, as shown in Figure 5.



max dirty mb	32	32	32	32	32	32	32	32	32	32	64	128	128
max readahead whole mb	2	2	2	2	2	2	2	0	1	2	2	2	1
max readahead mb	40	40	40	40	0	64	128	40	40	40	40	40	128
stripe size	1	4	64	128	1	1	1	1	1	1	1	1	4

Figure 5: Effects of varying some Lustre tunables on read performance (MB/s)

These are TestDFSIO comparisons of Hadoop on Lustre 1.8.0 using InfiniBand. The files were fully striped. The first column represents the Lustre default values for the parameters that we adjusted. Increasing the stripe size to 4MB improved both read and write performance. Beyond this value, read performance continued to increase, but write performance suffered considerably. Increasing the readahead buffer also improved read performance. Increasing the write cache size to 128MB improved write performance. Interestingly, combining the best individual tunings (4MB stripes, 128MB max_dirty, and 128MB max_readahead) yielded the best write performance by far, but only average read performance.

These results might provide direction for further investigation into tuning Lustre for a particular Map/Reduce workload.

Raw File System Copy

We also ran tests to measure the time needed to transfer a 1GB file to/from a local disk. To access data for analysis in Hadoop, input data must be transferred into HDFS (or a special input module can be written). Similarly, to use the output results of a Hadoop job, data must typically be made available to post-processing (e.g. visualization) software. To assess the overall job run time, this transfer time should be included. Generally, for Lustre, the transfer time is zero (0) because most secondary data processing/visualization assumes that a POSIX-compliant, distributed file system is in use. In Figure 6, we measure the data transfer time into and out of HDFS (3x replication, 1 GigE) and Lustre (IB). This is not a fair comparison with Lustre for two reasons: the values are measured on different networks, but also the practical value for Lustre in this situation would likely be zero (0), as described above. Transfer times to and from local disk are included for Lustre simply for comparison.

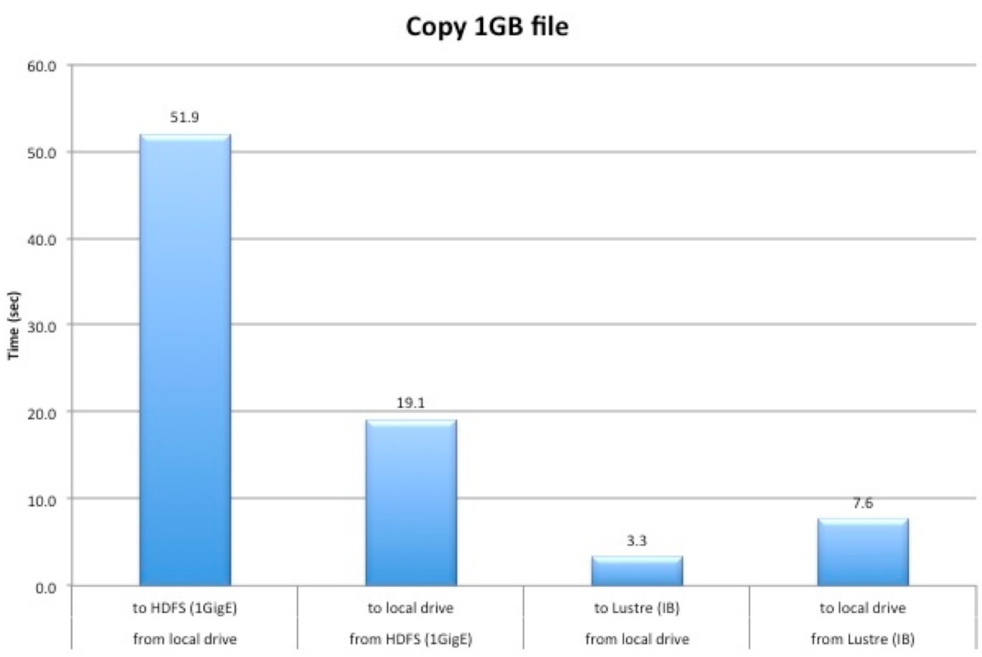


Figure 6: Time to transfer 1GB file to/from a local disk

The time to copy data into HDFS is large due to data replication. Reads from HDFS are faster, but still significantly slower than reads from Lustre, potentially due to caching and optimized network usage. Note the Lustre data most likely reflects local drive limits (the "other side" of the copy), not Lustre limits.

Second-Order Effects

Cluster Efficiency

By decoupling data storage from the nodes that perform computation on that data, we introduce flexibility in the assignment of Map and Reduce tasks for a Hadoop job. In an HDFS cluster, the number and location of Mapper tasks for a specific job is fixed by the distribution of the input data. If the data exists on only 30 nodes out of a 1000-node HDFS cluster, then only 30 Mapping tasks can run simultaneously, leaving most of the cluster idle. (While the data can be transferred to other nodes over the network, this conflicts with the "local compute" design; HDFS is not designed to distribute data over the network quickly and efficiently.) Alternatively, using Lustre as the backend file system for a Hadoop job permits complete flexibility in assigning Mapper tasks. All available nodes can be used for the same job, without incurring a network penalty. The input splits are equally flexible; each Mapper task could operate on 128MB of data or 4MB of data with, potentially, identical efficiency. In Lustre, the data can be moved efficiently to any compute resources that are available; thus node utilization can be maintained near 100% independently of the job.

Network Efficiency

Mapper tasks tend to finish at different times. Reduce tasks can only begin when the final Mapper task has finished writing its data. With Lustre, since the full bandwidth of the network and an entire set of drives are available to any compute node doing active I/O, the average time to write Mapper output should decrease. This efficiency means that the final Mapper task will almost always be written more quickly with Lustre than with HDFS, enabling an earlier start for the Reduce tasks - even on jobs that are mostly gated by CPU.

Reliability and Failure Modes

Hadoop is designed with the expectation that hardware failures (compute nodes and drives) will occur. HDFS uses data replication to ensure data availability in the event of a node failure. HDFS-RAID changes this design marginally, providing the ability to trade off some data replication for parity calculations instead. Lustre, by contrast, is designed to separate and layer failure modes using RAID disk drive units, failover server pairs, and diskless clients. The concentration of drives onto a small number of servers allows larger-scale, specialized safety/recovery RAID hardware to be used, resulting in higher data availability with fewer drives. With RAID-6, all data continues to be available, even in the event of two drives failing; this is the same level of data redundancy that 3x replication provides.

Hadoop provides resilience in the face of data center failures. This is not a part of Lustre's fundamental design, although it can be achieved for Lustre by replicating the Lustre file system via an external tool. Replication, of course, affects storage sizing requirements.

Cost Considerations

We next compare the approximate theoretical cost of two HPC Map/Reduce clusters, one based on HDFS with 1 GigE and one based on Lustre with IB. We have chosen to compare costs on different networks because our experiments have shown that using HDFS with IPoIB does not provide a significant performance benefit. HDFS is not highly network dependent, by design (e.g. we saw no performance gain in our sort benchmark). Lustre is highly network sensitive and benefits greatly from using IB instead of 1 GigE. Therefore, we priced a more expensive IB network into our theoretical Lustre configuration below, but did not add this cost to the HDFS cluster.

The clusters are designed to achieve similar performance on both systems. In our test cases, we have seen that the performance of Map/Reduce on Lustre is approximately three times faster than Map/Reduce on HDFS. Therefore, to achieve performance that equals Lustre, an HDFS cluster would require three times more compute nodes.

We conservatively size the HDFS node count at twice the Lustre node count, given that the relative performance will be task-dependent. Note that in practice, based on the Top 500 clusters[6] , 1 GigE systems usually produce LINPACK results with about 45-50% efficiency whereas QDR IB usually reaches 90-95%. These are not Map/Reduce jobs; this is the effect of the network fabric alone. So, from this perspective, choosing a 2x node multiplier for the 1 GigE HDFS system versus an IB Lustre system is conservative. Obviously, the performance of these two systems is highly dependent on the application being run and the I/O pattern.

The costs listed in Table 1 are approximate and provided for illustrative purposes. Of course, the relative costs will vary based on significant assumptions; nevertheless, this comparison shows that a higher-performance backend may, in practice, also be more cost-effective.

For estimated costs, we will use list prices from the Dell Computer website. (This is a "build-it-yourself" cluster with no third-party integration and uses free, open-source packages.) For all the compute nodes, we choose a Dell 1 RU server with 2x Xeon 5600@2.66 GHz CPUs (the most economical SKU) and 12GB RAM; this configuration lists for slightly more than $7,500.

6 J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software", Technical Report CS-89-85, Electrical Engineering and Computer Science Department, University of Tennessee, 2010.

The Lustre cluster will have 100 compute nodes, in 4 racks. For a non-blocking IB fabric, we will use nine 36-port switches (Mellanox 5030s at $6,500 each), and 178 IB cables. For the Lustre MDS, we will use a pair of PowerEdge R610s ($7,200), for failover, a PowerVault MD3220 ($18,500) with IB cards for a total of $34,000. For each OSS, we will use a PowerEdge R710 ($9,000) and three 30-TB PowerVault MD1000s ($14,000), with IB, for $52,000 each. A 4xQDR InfiniBand link can deliver 4GB/s point to point (and each of the two OSSs should be able to deliver ~4.5GB/s), for an average of 80MB/s throughput to each client. A 1.4x RAID-6 overhead (because of the drive configuration) gives 128TB of usable storage for the pair of OSSs.

The HDFS cluster will have 200 nodes, in 6 racks. A non-blocking 1 GigE fabric requires about 3 ports per node (as a rule-of-thumb), for twelve 48-port switches ($4,000) and cables. We will use the Hadoop standard 3x replication and install three times the amount of storage space in this system to equal the storage capacity of the Lustre cluster.

	Lustre / IB Cluster			HDFS / 1 GigE Cluster		
	Count	Price	Subtotal	Count	Price	Subtotal
Nodes	100	$7,500	$750,000	200	$7,500	$1,500,000
Switches	9	$6,500	$58,500	12	$4,000	$48,000
Cables	178	$100	$17,800	450	$10	$4,500
OSS	2	$52,000	$104,000	0	---	---
Storage	128TB	---	---	384TB	$100	$38,400
MDS	1	$34,000	$34,000	0	---	---
Racks	4	$8,000	$32,000	6	$8,000	$48,000
Total			**$996,300**			**$1,638,900**

Table 1: Sample HPC cluster costs (Lustre v. HDFS)

As shown in Table 1, the client node count dominates the overall cost of the cluster. Comparing the relative costs of the two theoretical clusters, a performance-normalized Lustre/IB cluster is less expensive than an equivalently performing HDFS/1 GigE cluster.

Secondly, the cost of running a larger cluster over time is significant. Over a given period, the HDFS system would cost at least twice as much to run as the Lustre solution, considering power, cooling, and general maintenance expenses.

Other cost factors include:

- Cluster utilization efficiency
- Data transfer (copy) time
- Necessity of maintaining a second cluster (i.e. both a POSIX-compliant, clustered file system for traditional HPC jobs and an HDFS cluster dedicated to Map/Reduce)

Conclusions

The Hadoop Distributed File System is based on fundamental design factors that are appropriate for certain environments, but that do not necessarily align well with HPC's performance-oriented requirements. In HPC environments, the HDFS design can result in scalability and performance limitations. Replacing HDFS with Lustre as the underlying file system for Hadoop compute clusters in HPC environments can, theoretically, result in significant improvements in system performance and cluster efficiency, or potentially offer a lower overall system cost. As shown in this paper, practical measurements on a small cluster support these conclusions. However, comparative testing on a much larger scale should be undertaken in the future. Similarly, relative performance is expected to vary significantly based on a particular task's input/output pattern; we highly recommend prototyping the performance for a cluster's anticipated tasks, before selecting a backend file system.

The choice of backing file system for any Map/Reduce cluster should depend on the task profile, data usage patterns, data safety concerns, and the desire to integrate other HPC algorithms or data stores. While there is no universal best choice, with increasing scale and performance requirements a careful study of alternatives becomes more critical.

About Xyratex

Xyratex is a leading provider of enterprise class data storage subsystems and hard disk drive capital equipment. The Networked Storage Solutions division designs and manufactures a range of advanced, scalable data storage solutions for the Original Equipment Manufacturer (OEM) community. As the largest capital equipment supplier to the industry, the Storage Infrastructure division enables disk drive manufacturers and their component suppliers to meet today's technology and productivity requirements. Xyratex has over 25 years of experience in research and development relating to disk drives, storage systems and manufacturing process technology.

Founded in 1994 in an MBO from IBM, and with headquarters in the UK, Xyratex has an established global base with R&D and operational facilities in Europe, the United States and South East Asia.

Xyratex Headquarters
Langstone Road
Havant
Hampshire PO9 1SA
United Kingdom
T +44 (0)23 9249 6000
F +44 (0)23 9249 2284

Principal US Office
46831 Lakeview Blvd.
Fremont, CA 94538
USA
T +1 510 687 5200
F +1 510 687 5399

www.xyratex.com

BSI REGISTERED UKAS ENVIRONMENTAL MANAGEMENT 003

ISO 14001: 2004 Cert No. EMS91560

xyratex