On the Use of Shared Storage in Shared-Nothing Environments

Krish K.R[†], Aleksandr Khasymski[†], Guanying Wang[†], Ali R. Butt[†], Gaurav Makkar[‡]

[†]*Dept. of Computer Science, Virginia Tech*; [‡]*NetApp Inc.*
Email: {kris,khasymskia,wanggy,butta}@cs.vt.edu; {Gaurav.Makkar}@netapp.com

***Abstract*—Shared-nothing environments, exemplified by systems such as MapReduce and Hadoop, employ node-local storage to achieve high scalability. The exponential growth in application datasets, however, demands ever higher I/O throughput and disk capacity. Simply equipping individual nodes in a Hadoop cluster with more disks is not scalable as it: increases the per-node cost, increases the probability of storage failure at the node, and worsens node failure recovery times. To this end, we propose dividing a Hadoop rack into several (small) sub-racks, and consolidating disks of a sub-rack's compute nodes into a separate shared Localized Storage Node (LSN) within the sub-rack. Such a shared LSN is easier to manage and provision, and can offer an economically better solution by employing overall fewer disks at the LSN than the total of the sub-rack's individual nodes, while still achieving high I/O performance.**

In this paper, we provide a quantitative study on the impact of shared storage in Hadoop clusters. We utilize several typical Hadoop applications and test them on a medium-sized cluster and via simulations. Our evaluation shows that: (i) the staggered workload allows our design to support the same number of compute nodes at a comparable or better throughput using fewer total disks than in the node-local case, thus providing more efficient resource utilization; (ii) the impact of lost locality can be mitigated by better provisioning the LSN-node network interconnect and the number of disks in an LSN; and (iii) the consolidation of disks into an LSN is a viable and efficient alternative to the extant node-local storage design. Finally, we show that LSN-based design can deliver up to 39% performance improvement over standard Hadoop.

I. Introduction

In recent years, enterprises have been increasingly adopting the MapReduce [9] model — and its publicly available open-source implementation, Hadoop [6] — for data processing and analytics. The main attraction of MapReduce is its computational model, which hides the complex system-level details of fault tolerance, replication, and data management from the application programmer. This allows for large data-intensive applications to be transparently parallelized on large clusters.

Hadoop clusters are typically built using commodity machines (nodes) and employ a shared-nothing architecture. In Hadoop's context, shared nothing implies that local resources, such as CPU, memory, and disks are not shared across nodes, even if they are in the same rack. Each Hadoop node typically serves as both a compute node, executing the tasks, and a storage node, storing the associated data on local disks. Any interactions between nodes occur explicitly and only during the shuffle phase of MapReduce. The application tasks spend the majority of their time only using node-local resources and consequently the system can achieve very-high scalability. This shared-nothing property also provides simplified overall application semantics; a node failure does not affect other nodes, and the failed node can be easily replaced by assigning its tasks to a different node. On the flip side, isolating local resources implies that idle resources at one node cannot be used to serve the needs of another node experiencing a workload spike. This is critical, as although ideally MapReduce workloads should be equally divided among the participating nodes; in reality, skew in data/task assignment results in a load imbalance between the nodes [3]. Thus, the shared-nothing model leads to inefficiencies and resource-fragmentation as resources can be under- or over-provisioned depending on the workload assigned to the node, even when enough resources are available in the system to handle the workload.

Consider the case of provisioning a node for a desired disk throughput. One solution used in large-scale setups is equipping each node with more disks and stripping data across them to handle load spikes [7]. However, this not only results in costly over-provisioning, but also increases node failure recovery times and complicates fault tolerance semantics.

In this paper, we posit that aggregating and sharing resources across nodes can produce an efficient resource allocation in an otherwise shared-nothing Hadoop cluster. To this end, we divide a Hadoop cluster rack into several sub-racks, and consolidate disks of a sub-rack's compute nodes into a separate shared Localized Storage Node (LSN) within the sub-rack. The scope of a single LSN can range from serving a few nodes to perhaps a complete Hadoop cluster rack.

A key observation that makes this approach viable is that in typical Hadoop clusters, accesses to disks are often staggered in time because of the mix of different types of jobs and data skew across nodes. This, coupled with the bursty node workload, implies that contention at the LSN from its associated nodes is expected to be low. Therefore, by simply repurposing a sub-rack's node-local disks in an LSN, each node can receive a higher instantaneous I/O throughput provided by the larger number of disks in the LSN. Conversely, the LSN can service its associated nodes at the default I/O throughput (experienced by nodes using their local disks only) with fewer number of disks at the LSN. We note that we do not argue for provisioning LSNs in addition to the node-local disks, rather placing some or all of the disks from a sub-rack's nodes at

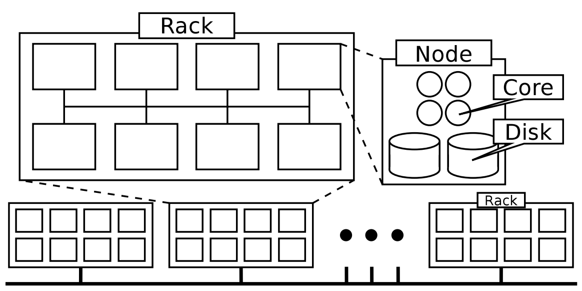

Fig. 1. Standard Hadoop cluster architecture.

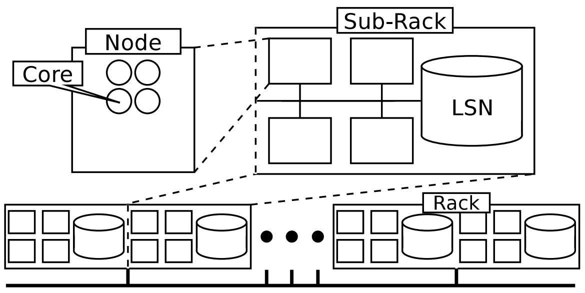

Fig. 2. Hadoop architecture using an LSN.

their LSN. Of course, moving the disks away from a node and into a shared LSN results in loss of data locality, so achieving higher I/O throughput depends on appropriate provisioning of both disks and network bandwidth to the LSN. Our design, thus, *provides a practical control knob for realizing a desired performance-cost operating point for a Hadoop cluster*.

Another advantage of LSN-based design is that it decouples storage and compute provisioning, and allows for scaling up storage to meet the demands of big data applications by simply provisioning more disks at the LSN. Consolidating data into fewer high-density nodes opens the door for a myriad of global decisions and optimizations, such as deduplication, compression, and snap-shot generation. Standard enterprise fault tolerance techniques, such as RAID-5 and RAID-6, can also be employed more easily in our LSN-based design.

II. Integrating Shared Storage In Hadoop

In this section, we motivate our design of consolidating disks from individual Hadoop nodes into a local shared storage, and then outline several alternative shared-storage designs.

A. Hadoop Architecture

A typical Hadoop cluster topology organizes nodes into racks as shown in Figure 1. In addition to a MapReduce runtime, Hadoop also includes the Hadoop Distributed File System (HDFS) that is based on GFS [10]. HDFS consists of a master data management component, *NameNode*, and worker components called *DataNodes*, which also run the compute managers *JobTracker* and *TaskTrackers*, respectively. HDFS divides the data into blocks (chunks) and distributes them across all DataNodes in the cluster. Moreover, the system typically maintains three replicas of each data block, two placed within the same rack and one outside.

B. Rationale and Motivation

Application datasets continue to grow at unprecedented rates. To keep up with this trend, the per-node disk capacity on Hadoop clusters is increasing rapidly, e.g., from two 80 GB disks in the original MapReduce deployment [10] to four and (even eight) 3 TB disks [7] in modern Hadoop setups. This raises several issues about the viability of using node-local storage for all data. First, simply adding more disks to local nodes increases the chance of some disks failing, and reduces the already typically low Mean Time Between Failures (MTBF) of a Hadoop node. Second, when using commodity Hadoop hardware, a significant time and bandwidth resources are spent on recreating replicas after node or disk failures. In contrast, Enterprise storage solutions have lower MTBF than commodity hardware clusters and ensure lower failure rate for the data [5], [4]. Third, provisioning all the storage needs of a node locally prevents use of advanced solutions such as the use of SSDs, as current price-points make it economically impractical to deploy such approaches at all the nodes. Finally, we argue that following the conventional wisdom of treating data-locality as the only design constraint in Hadoop clusters, results in a suboptimal solution, both in terms of performance and efficient utilization of resources. Factors such as storage utilization can no longer be ignored in the face of growing datasets.

Consider the following scenario. If a single task utilizes a single disk for 5% of its execution time, about 20 tasks are needed to fully utilize a single disk, given that tasks are all staggered so they do not compete with each other. If a node has 4 local disks, it takes about 80 tasks running concurrently to fully utilize all disks on the node. Moreover, if the tasks are not uniformly assigned to different nodes, which is typical, there will be a skew in the load with some nodes experiencing I/O bottlenecks, while others with idle disks. A better solution is to sacrifice some locality and consolidate disks into a localized storage node (LSN), which can service multiple nodes and thus achieve better disk utilization.

To this end, we propose consolidating the disks from a small number of compute nodes into an LSN, which yields higher average disk utilization and simplified cluster provisioning.

C. Alternate Storage Sharing Scenarios in Hadoop

A consolidated shared storage system can reside at different levels of the Hadoop architecture. In the following, we present three potential alternative scenarios for sharing storage in Hadoop.

1) Naive Storage Consolidation: A first cut design is to take all MapReduce related data and move it to a single consolidated storage outside the entire Hadoop compute cluster, and provision a very high bandwidth link between the compute nodes and the storage system. Such a setup is often deployed to connect a cluster file system to a supercomputer. However, in this configuration a typical large-scale data-intensive Hadoop

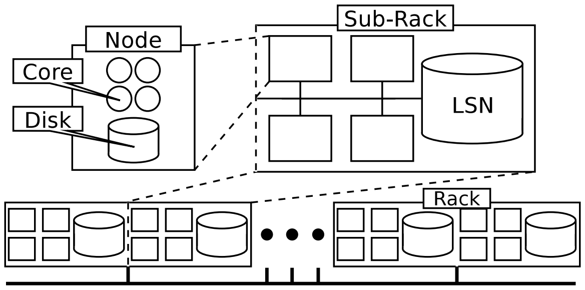

Fig. 3. Hadoop architecture using a hybrid storage design comprising of a small node-local disk for shuffle data and an LSN for supporting HDFS.

application would create an almost constant high-volume I/O flood to the storage system, which would quickly saturate the storage connection link and become a bottleneck. Moreover, aggregating storage cluster-wide would require a sophisticated cluster file system that treats the storage nodes as an integrated unit. This in turn would entail complexity in managing failures and providing high performance. Consequently, such a design goes against the very spirit of the MapReduce model that achieves unprecedented scalability by treating the cluster resources as loosely coupled with data stored locally, and that are readily replaceable.

2) Localized Storage Consolidation: The main bottleneck in the previous case is the interconnect between the global shared storage and Hadoop nodes. In our next design, shown in Figure 2, we limit the number of compute nodes that share a storage system, i.e., a sub-rack whose size range from a fraction of a rack to perhaps a complete rack. We refer to the shared storage as Localized Storage Node (LSN). The intuition behind this local consolidation approach is that it avoids the bandwidth bottlenecks by limiting the sharing to a few nodes instead of the whole cluster. All the disks from the sub-rack's compute nodes are consolidated into a corresponding LSN for the sub-rack. The LSN supports both their HDFS and shuffle data for the sub-rack.

In this configuration, map tasks no longer have node-level locality and must retrieve data from the corresponding LSN in the sub-rack. However, since data is now striped across a larger number of disks at the LSN than those of a single node, the LSN can provide much higher I/O throughput, which can mitigate the impact of lost locality. Moreover, since only a small number of nodes share an LSN, only the inter-rack interconnect is used for accessing data, and multiple sets of nodes (in different racks) can interact with their LSNs simultaneously, avoiding a global bottleneck.

3) Hybrid Storage Consolidation: One limitation of the previous design is that each compute node requires at least one local disk to run its operating system, and thus makes it impractical to remove all disks from a node to its associated LSN. The key insight of our next design, shown in Figure 3, is to store shuffle data, which is not replicated and usually consumed shortly after it is generated, on the node-local disk. Thus, we design a hybrid setup where the LSN stores HDFS data for a sub-rack of nodes, while a local node disk stores shuffle data and OS files required to run the node.

	Network		Data Storage		
Conf	N_1	N_2	D_1	D_2	D_3
Nodes	1 Gbps	10 Gbps	1 disk		
LSN	1 Gbps	10 Gbps	1 disk	3 disks	5 disks
Speedup	1	10×	1	3×	5×

TABLE I
NETWORK AND DISK CONFIGURATIONS IN THE TESTBED.

An extra advantage of the hybrid approach is that it paves the way for economically incorporating SSDs in the Hadoop architecture. For instance, the node-local disks can be replaced by (low-capacity) SSD devices for holding the OS and serving as a buffer for in-memory shuffle data. Given the good random I/O (especially read) performance of SSDs [1], handling shuffle data would be a well-matched use-case for them. This is also advocated by recent work on the importance of memory-locality rather than disk-locality in Hadoop [2].

III. Evaluation

In this section, we present the evaluation of the hybrid localized shared storage design in Hadoop. We compare a *baseline* Hadoop to LSN-based one.

A. Tests Using a Real Cluster

Our first set of tests explore the impact of LSN on Hadoop performance using a real cluster.

1) Experimental Setup: Our testbed consists of a master node and 21 worker nodes serviced by three LSNs. The nodes have two 2.8 GHz quad-core Intel Xeon processors, 8 GB of RAM, and 1 SATA disk. The LSN nodes are identical to the rest of the nodes, but contain 5 SATA disks. The disks are Seagate Barracuda ES.2 7200 RPM with 500 GB capacity. The machines are connected to a dedicated Gigabit switch via 1 Gbps links as well as a dedicated InfiniBand switch via 10 Gbps links. We ran the three basic benchmarks: *TeraGen* with 1 mapper per compute node, *Grep* and *TeraSort* each with 16 mappers and 2 reducer per compute node. *TeraGen* generates 1 GB of data per worker node, which is the input for *Grep* and *TeraGen*.

2) LSN Performance: For our next experiment, we used 15 worker nodes to compare the performance of standard Hadoop (*baseline*) to that of LSN-based Hadoop using different network and storage provisioning. Figure 4 shows the results for six hybrid configurations with 3 LSNs, one per 5 worker nodes, and two standard configurations, where (N_x, D_y) refers to N_x and D_y in Table I. As a first point of comparison, we test $LSN(N_1, D_1)$, which consolidates all the HDFS storage onto a single remote disk per LSN, without changing the network bandwidth. Not surprisingly, going from fifteen local disks, to three remote disk, slowed *TeraGen*, *grep* and *TeraSort* by 2.45×, 2.55× and 1.50×, respectively, compared to $Baseline(N_1, D_1)$. The three disks do not provide enough bandwidth to keep up with the fifteen compute nodes and hence becomes a bottleneck.

Next, we test $LSN(N_2, D_1)$, which increases the network throughput available to each node, but still has only one disk

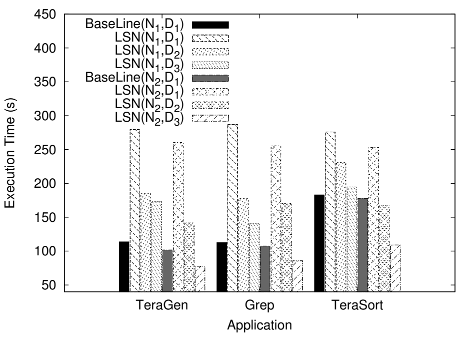

Fig. 4. Comparison of Hadoop (*baseline*) with LSN-based configurations.

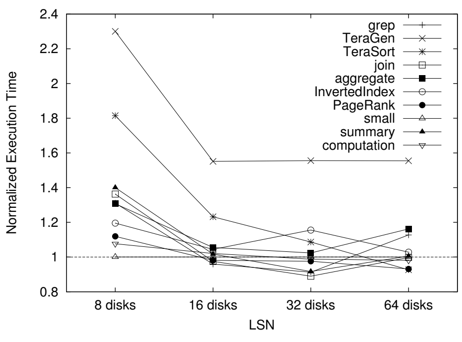

Fig. 5. Performance of *baseline* Hadoop and LSN with different number of disks in LSN. Network speed is fixed at 40 Gbps.

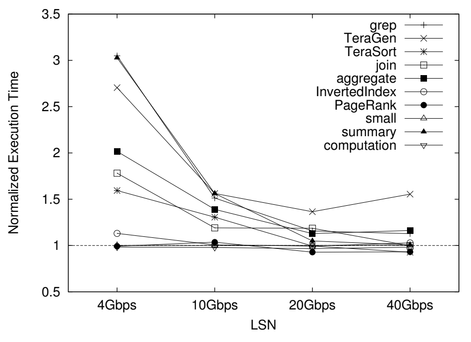

Fig. 6. Performance of *baseline* Hadoop and LSN with different network bandwidth to LSN. The number of disks at LSN is fixed at 64.

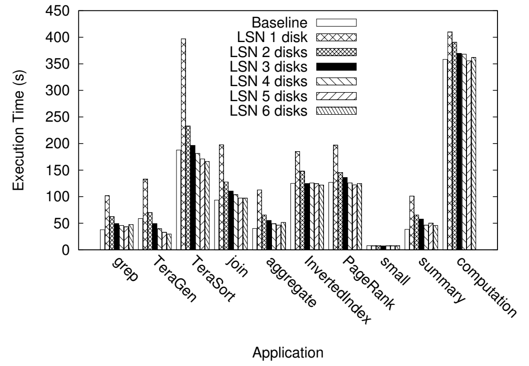

Fig. 7. Performance of *baseline* Hadoop and LSN with different number of disks in LSN. The network speed is fixed at 4 Gbps.

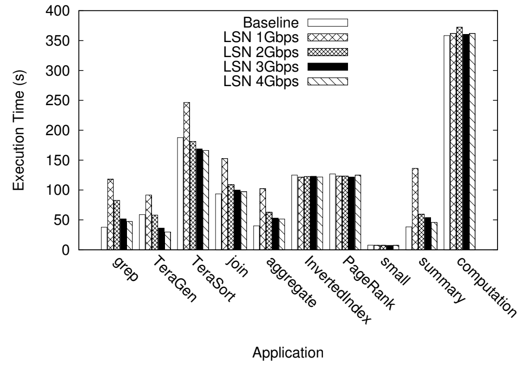

Fig. 8. Performance of *baseline* Hadoop and LSN with different network bandwidth to LSN. The number of disks at the LSN is fixed at 6.

per LSN. The higher network bandwidth does not improve the performance and *TeraGen*, *grep* and *TeraSort* execute 2.55×, 2.37× and 1.42× slower than $Baseline(N_2, D_1)$, respectively. In the next test, $LSN(N_1, D_2)$, we increase the number of disks, but not the network bandwidth. We see that the addition of disks improves the performance, as this configuration performed better than $LSN(N_1, D_1)$ with a performance improvement of 1.5×, 1.6× and 1.2× in *TeraGen*, *grep* and *TeraSort*, respectively. The $LSN(N_1, D_3)$ configuration uses even more disks, however, the network is a bottleneck and hence there is little extra benefit from adding the disks. Finally, we test better provisioning both network and storage, which is the intended deployment of our system. $LSN(N_2, D_3)$ sees a performance increase by 24%, 20% and 39% in *TeraGen*, *grep*, and *TeraSort*, respectively, compared to $BaseLine(N_2, D_1)$.

These results show that balancing shared disk provisioning with an adequate network throughput to the LSN can perform better than the shared-nothing *baseline* Hadoop.

B. Simulation Results

1) Simulating Hadoop Clusters: In this set of experiments, we use simulation to study the impact of our LSN-based Hadoop in detail. Table II lists the applications and summarizes parameters therein, such as the input and output data size, the number of mappers and reducers, which we simulate.

There is ample previous research done on modeling and simulation of MapReduce workloads and setups [14], [17], [19], [16], [13], [11], [12], which we leverage. We choose our MRPerf [19] discrete event Hadoop simulator, as it has been previously used to study impact of data locality, alternative network topologies, and failure [19], [18]. The simulator provides us with means to investigate the performance impact of system features such as node, rack, and network configurations, disk parameters and performance, data layout, and application I/O characteristics — which we want to explore in the context of LSN design.

The MRPerf simulator takes as input the topology of a cluster, the parameters of a job, and a data layout, and produces detailed simulation results about how the job would behave on the specified cluster configuration. In this work, we extended

Application	Map		Reduce	Number	
	Input	Output	Output	Mapper	Reducer
Grep	10 GB	1 MB	1 MB	160	1
TeraGen	0 KB	10 GB	–	40	–
TeraSort	10 GB	10 GB	15 GB	160	40
Join	10 GB	1 GB	10 MB	160	40
Aggregate	10 GB	100 MB	10 MB	160	10
Inverted Index	1 GB	10 GB	100 MB	40	40
PageRank	1 GB	10 GB	1 GB	40	40
Small	100 KB	1 MB	10 KB	4	1
Summary	10 GB	10 MB	10 KB	160	1
Compute	1 GB	10 GB	100 MB	40	40

TABLE II
REPRESENTATIVE MAPREDUCE (HADOOP) APPLICATIONS USED IN OUR STUDY. THE PARAMETERS SHOWN ARE THE VALUES USED IN OUR SIMULATIONS. FOR *TeraGen* THE LISTED MAP COST IS WITH RESPECT TO THE OUTPUT.

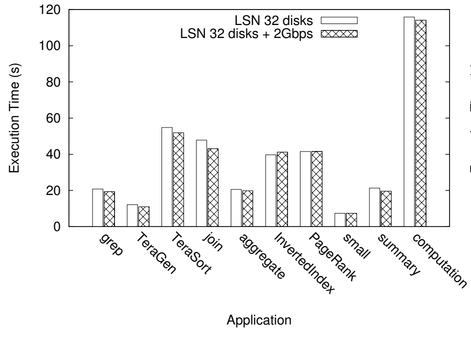

Fig. 9. LSN performance with Hadoop nodes equipped 2 Gbps links.

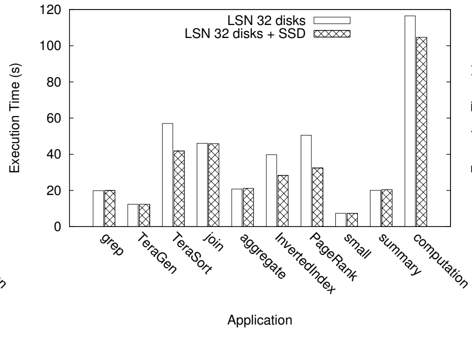

Fig. 10. LSN performance with Hadoop nodes equipped with SSDs.

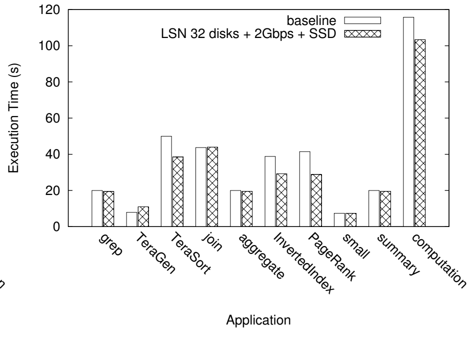

Fig. 11. *baseline* performance compared to LSN with nodes equipped with SSDs and 2 Gbps links.

the simulator to support our application-oriented evaluation. The simulations use deterministic traces across multiple runs.

We focus on simulating a single sub-rack and analyze in detail different LSN designs choices and their effect on performance. We believe that the conclusion we draw will generalize to larger clusters comprising multiple sub-racks, each with its own LSN. This is true especially when a better provisioned communication channel is used between nodes and their associated LSNs to avoid the interconnect contention due to node-LSN traffic of different LSNs.

2) 20-node LSN Simulation: We consider a topology with 20 nodes and 1 LSN. Each of the 20 nodes has 8 cores and 4 disks and is connected via 1 Gbps links. In the LSN case, we aggregate up to 64 disks, leaving one disk at each node, and connect it to the switch via a 40 Gbps link. In some of the cases, we also increase each node's interconnect to 2 Gbps links and equip them with SSDs. All experiments are run with 8 map slots and 4 reduce slots.

In Figure 5, we change the number of disks provisioned at the LSN and measure the execution time of each application normalized to the *baseline*. In this case, the LSN's network is set to maximum throughput at 40 Gbps to make sure it does not become a bottleneck. The performance numbers of the $LSN(N_{40}D_{16})$ configuration, which are within 5.5% as compared to the *baseline* Hadoop for eight out of ten applications, illustrate the efficiency of our disk aggregation technique. In this 20 node cluster, we are able to efficiently utilize 55% fewer disks ($20+16=36$ disks in $LSN(N_{40}D_{16}$ compared to $20*4 = 80$ disks in *baseline*) to achieve comparable performance for the studied applications. The two applications, *TeraGen* and *TeraSort*, which are very output-heavy see a 55% and 23% slowdown, respectively. In both these cases, LSN becomes a bottleneck, as it is unable to keep up with the workload.

Next we investigate the effect of network bandwidth on application performance. For this experiment, we set the number of disks to 64. The results are plotted in Figure 6. We see that a 4 Gbps connection is sufficient to support half of the applications, i.e., the ones that do not generate large amount of data and 20 Gbps is enough to bring performance of all applications except *TeraGen* to within 18.6%.

Next, we set up a simulation topology with 5 nodes and 1 LSN to simulate a smaller LSN to node ratio and again study the effect of different design choices. All nodes are connected through a single switch. Connection speed for each node and LSN is 1 Gbps and 4 Gbps, respectively. Each of the five nodes has 8 cores and 2 disks, while the LSN has 6 disks. We run ten Hadoop applications and record the results. Each node is configured with 8 map slots and 4 reduce slots.

The first test studies performance under varying number of disks provisioned at the LSN. Figure 7 shows the results. The Figure shows several aspects of design trade-offs for LSN. First, LSN with 4 disks can match performance of *baseline* Hadoop within 3.7%, on average, and there is almost no benefit of adding more disks. This means LSN provide saving of a disk. (9 disks in LSN versus 10 in *baseline*). Second, output heavy jobs like *TeraGen* see a significant performance boost, 33% as seen by *TeraGen*, compared to *baseline* provided mainly by the reduced number of replications. Moreover, LSN can load balance between multiple nodes, and achieve high overall performance. Third, read heavy workloads, such as *Grep*, *Aggregate*, and *Summary* exhibit more uniform access patterns to local disk and as a consequence experience a small slowdown, 18.7% on average, when running on the LSN. This is because aggregating the access at the LSN does not provide an additional benefit. Finally, the rest of the applications are within 4.6% of the *baseline*.

Next, we vary the bandwidth available at the LSN from 1 Gbps to 4 Gbps, and observe the performance impact. Figure 8 summarizes the results. The four applications — *InvertedIndex*, *PageRank*, *Small*, and *Computation* — that do not consume or generate large amounts of data, but rather are CPU intensive or operate on large amount of intermediate data, see no benefit from increasing the LSN's network. In contrast, the rest of the applications, which do input/output experience a significant slowdown from the *baseline* with 1 Gbps link at the LSN. Provisioning 3 Gbps at the LSN, however, is enough to handle the client workload with a performance overhead within 5.7%, on average.

3) Better Provisioned Local Nodes: So far we have examined various provisioning scenarios for the LSN. In the next set of experiments, we take a look at several design options at the node side. To ensure that for these experiments the LSN is not a bottleneck, we provision it with 32 disks and a 20 Gbps network and set the map and reduce slots to four.

First, we examine the impact of increased local bandwidth of each node as seen in Figure 9. A 2 Gbps link produces a 4.0% speedup on average, most noticeably for *Join*, which benefits from the extra bandwidth for both its heavy HDFS and shuffle traffic and achieves a 9.9% speedup.

Our hybrid LSN approach significantly decreases the number and size of disks needed to be provisioned on each node, which lets us optimize each node by replacing its hard disk with an economically viable small-size SSD. The only workload related data that needs to be stored at the nodes is shuffle data. Shuffle data tends to create I/O workloads that mostly consist of random accesses [9]. The shuffling works in a pulling model, where consuming reducers proactively retrieves data from producing mappers [8]. Hence, the workload is characterized by sequential writes and random reads, which is a good match for the excellent random read performance of SSDs. The results of adding an SSD to each node are shown in Figure 10. *TeraSort*, *InvertedIndex*, *PageRank*, and *Computation*, all of which process a lot of intermediate data, get a significant performance boost (25.4% on average).

Finally, we combine the node-side optimizations of using an SSD and a faster link, and compare the performance to *baseline* Hadoop. The results are shown in Figure 11. The optimizations coupled together help us bridge the performance gap of even the most data intensive applications like *TeraGen* to 39.3% (from 53.7% without). The rest of the benchmarks achieve a 10.7% speedup in general and prove that our hybrid localized storage is a viable augmentation of the otherwise shared-nothing Hadoop architecture.

IV. Related Works

The work closest to ours is from Porter [15], which proposes consolidating Hadoop node disks into a SuperDataNode and running DataNode instances in virtual machines on that node. The goal of the SuperDataNode is to decouple storage from computation. The authors study basic Hadoop benchmarks using their SuperDataNode, but do not examine the underlying issues that cause the varying performance they encounter. While we share the concept of disk consolidation, our work is novel in its observation that resources are not well-utilized in a bursty workload environment due to Hadoop's shared-nothing architecture. More recently, Ganesh et al. [2] argue that neither disk locality nor remote memory access is good enough in data center computing. The authors argue for developing mechanisms that support memory locality, so applications achieve high throughput by mostly employing in-memory accesses. Achieving memory locality is complementary to our work.

V. Conclusion

In this paper, we revisit the cluster architecture of Hadoop to better provision per-node storage resources in the face of growing application datasets. We observe that simply adding more disks to individual Hadoop nodes that often exhibit bursty workloads is not efficient. This approach results in low overall disk utilization, increases costs as well as the chances of node failures, and the large capacity elongates the time it would take to recreate a failed replica. Also, scaling storage coupled with compute adds extra cost of non-storage resources that are not necessarily required for I/O-intensive workloads, and reduces overall system efficiency. To this end, we study the impact of LSN on Hadoop application performance using a range of representative applications and configuration parameters. Our evaluation shows that a single LSN servicing 20 compute nodes can achieve performance within 5.5% of standard Hadoop on average, for eight out of ten applications studied, while using a little over half (55%) of the number of disks in the standard setup. Moreover, for a case where an LSN is used by 5 compute nodes, we observe up to 12% performance improvement while using 28% fewer disks.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX ATC*, pages 57–70, 2008.

[2] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Disk-locality in datacenter computing considered irrelevant. In *Proceedings of the 13th USENIX HotOS*, HotOS'13, pages 12–12, 2011.

[3] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. Pacman: Coordinated memory caching for parallel jobs. In *9th USENIX NSDI*, pages 14–14, 2012.

[4] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[5] Andy Konwinski and et.al . X-tracing Hadoop, 2008. Hadoop Summit.

[6] Apache Software Foundation. Hadoop, 2011. http://hadoop.apache.org/core/, accessed on Nov 6, 2012.

[7] D. Borthakur. Facebook has the world's largest hadoop cluster!, 2010. http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html.

[8] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *7th USENIX NSDI*, pages 21–21, 2010.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. USENIX OSDI*, pages 137–150, 2004.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SOSP*, 2003.

[11] S. Hammoud, M. Li, Y. Liu, N. Alham, and Z. Liu. Mrsim: A discrete event based mapreduce simulator. In *Fuzzy Systems and Knowledge Discovery (FSKD)*, volume 6, pages 2993 –2997, aug. 2010.

[12] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, pages 261–272, 2011.

[13] Y. Liu, M. Li, N. K. Alham, and S. Hammoud. Hsim: A mapreduce simulator in enabling cloud computing. *Future Generation Computer Systems*, (0):–, 2011.

[14] A. C. Murthy. Mumak: Map-Reduce Simulator. ASF JIRA MAPREDUCE-728, 2009.

[15] G. Porter. Decoupling storage and computation in hadoop with super-datanodes. *SIGOPS Oper. Syst. Rev.*, 44:41–46, April 2010.

[16] F. Teng, L. Yu, and F. Magoules. Simmapreduce: A simulator for modeling mapreduce framework. In *Multimedia and Ubiquitous Engineering (MUE), 2011 5th FTRA International Conference on*, pages 277 –282, june 2011.

[17] A. Verma, L. Cherkasova, and R. H. Campbell. Play it again, simmr! In *CLUSTER*, pages 253–261. IEEE, 2011.

[18] G. Wang, A. R. Butt, H. Monti, and K. Gupta. Towards synthesizing realistic workload traces for studying the Hadoop ecosystem. In *Proc. IEEE MASCOTS*, 2011.

[19] G. Wang, A. R. Butt, P. Pandey, and K. Gupta. A Simulation Approach to Evaluating Design Decisions in MapReduce Setups. In *Proc. IEEE MASCOTS*, pages 1–11, 2009.