Can High-Performance Interconnects Benefit Hadoop Distributed File System?

Sayantan Sur, Hao Wang, Jian Huang, Xiangyong Ouyang and Dhabaleswar K. Panda

Department of Computer Science and Engineering, The Ohio State University
{*surs, wangh, huangjia, ouyangx, panda*}*@cse.ohio-state.edu*

Abstract

During the past several years, the MapReduce computing model has emerged as a scalable model that is capable of processing petabytes of data. The Hadoop MapReduce framework, has enabled large scale Internet applications and has been adopted by many organizations. The Hadoop Distributed File System (HDFS) lies at the heart of the ecosystem of software. It was designed to operate and scale on commodity hardware such as cheap Linux machines connected with Gigabit Ethernet. The field of High-performance Computing (HPC) has been witnessing a transition to commodity clusters. Increasingly, networks such as InfiniBand and 10Gigabit Ethernet have become commoditized and available on motherboards and as low-cost PCI-Express devices. Software that drives InfiniBand and 10Gigabit Ethernet works on mainline Linux kernels. These interconnects provide high bandwidth along with low CPU utilization for network intensive operations. As the amounts of data processed by Internet applications reaches hundreds of petabytes, it is expected that the network performance will be a key component towards scaling data-centers. In this paper, we examine the impact of high-performance interconnects on HDFS. Our findings reveal that the impact is substantial. We observe up to 11%, 30% and 100% performance improvement for the sort, random write and sequential write benchmarks using magnetic disks (HDD). We also find that with the emerging trend of Solid State Drives, having a faster interconnect makes a larger impact as local I/O costs are reduced. We observe up to 48%, 59% and 219% improvement for the same benchmarks when SSD is used in combination with advanced interconnection networks and protocols.

I. Introduction

The MapReduce computing model has recently emerged as a viable model for processing petabytes of data. This model for processing data in large Internet warehouses, was first proposed by Google Inc [7]. The MapReduce model enables developers to write highly parallel codes without dealing with many intricate details of data distribution and fault tolerance. Most importantly, the model aims to be efficient on commodity clusters connected with Gigabit Ethernet. This model can handle both structured and unstructured data. Since the time when the MapReduce paper was published, Doug Cutting et. al. started developing Hadoop, which is an Open-source implementation of the MapReduce computing model. It is available from the Apache Software Foundation [27]. The Hadoop MapReduce software relies on the Hadoop Distributed File System (HDFS) as the underlying basis for providing data distribution and fault tolerance. Over time, HDFS has also become the underlying file system for the Hadoop database (HBase), that is an Open-source implementation of Google's BigTable [6]. The goal of HBase is to provide random, real time read/write access to large quantities of data, in the order of billions of rows and millions of columns.

The Hadoop project has gained widespread acceptance and is very widely used in many organizations around the world. As data gathering technologies (such as sensors) witness an explosion, it is expected that in the future, massive quantities of data in hundreds or thousands of petabytes will need to be processed to gain *insight* into patterns and trends. In order to process these large quantities of data, many more thousands of servers may be required. While the Hadoop framework has no fundamental scaling limitations, recently there has been some discussion about its efficiency. In particular, data-centers of the future cannot expand at the rate at which data storage and gathering capabilities are expanding. This is due to power limitations. Improving efficiency of the HDFS will have a significant impact on the design of future data-centers.

During the past decade, the field of High-performance Computing has been witnessing a transition to commodity clusters connected with modern interconnects such as InfiniBand and 10Gigabit Ethernet. Increasingly, InfiniBand has become commoditized and available on motherboards and as low-cost PCI-Express devices. Software that drives InfiniBand and 10Gigabit Ethernet also works on mainline Linux kernels. These interconnects provide not only high bandwidth (up to 32Gbps), and low latency (1μs-2μs), but also help server scalability by using very little CPU and reduced memory copies for network intensive operations. The popularity of InfiniBand as such, can be measured by

the fact that 42.6% of the compute clusters in the Top500 list [29] of most powerful supercomputers use InfiniBand. These clusters are very high on the efficiency metric, i.e. performance achieved compared to peak performance. Typical efficiencies of InfiniBand clusters range from 85%-95%.

As the amount of data processed by Internet applications reaches hundreds of petabytes, it is expected that the network performance will be a key component towards scaling data-centers. In this paper, we examine the impact of high-performance interconnects on HDFS. Our findings reveal that the impact is substantial. We observe up to 11%, 30% and 100% performance improvement for the sort, random write and sequential write benchmarks using magnetic disks (HDD). We also find that with the emerging trend of Solid State Drives, having a faster interconnect makes a larger impact as local I/O costs are reduced. We observe up to 48%, 59% and 219% improvement for the same benchmarks when SSD is used in combination with advanced interconnection networks and protocols.

The rest of the paper is organized as follows. In Section II, we provide an overview of the topics dealt with in this paper. In Section III, we show some of the benefits of modern interconnects. Experimental results and discussions are presented in Section IV. We discuss related work in Section V. We conclude the paper in Section VI.

II. Background

In this Section, we provide a "bottom-up" overview of networking and software components in a data-center that is interconnected using High-performance networks.

A. InfiniBand Overview

InfiniBand [2] is an industry standard switched fabric that is designed for interconnecting nodes in HEC clusters. It is a high-speed, general purpose I/O interconnect that is widely used by scientific computing centers world-wide. The recently released TOP500 rankings in November 2010 reveal that more than 42% of the computing systems use InfiniBand as their primary interconnect. The yearly growth rate of InfiniBand in the TOP500 systems is pegged at 30%, indicating a strong momentum in adoption. One of the main features of InfiniBand is Remote Direct Memory Access (RDMA). This feature allows software to remotely read memory contents of another remote process without any software involvement at the remote side. This feature is very powerful and can be used to implement high-performance communication protocols.

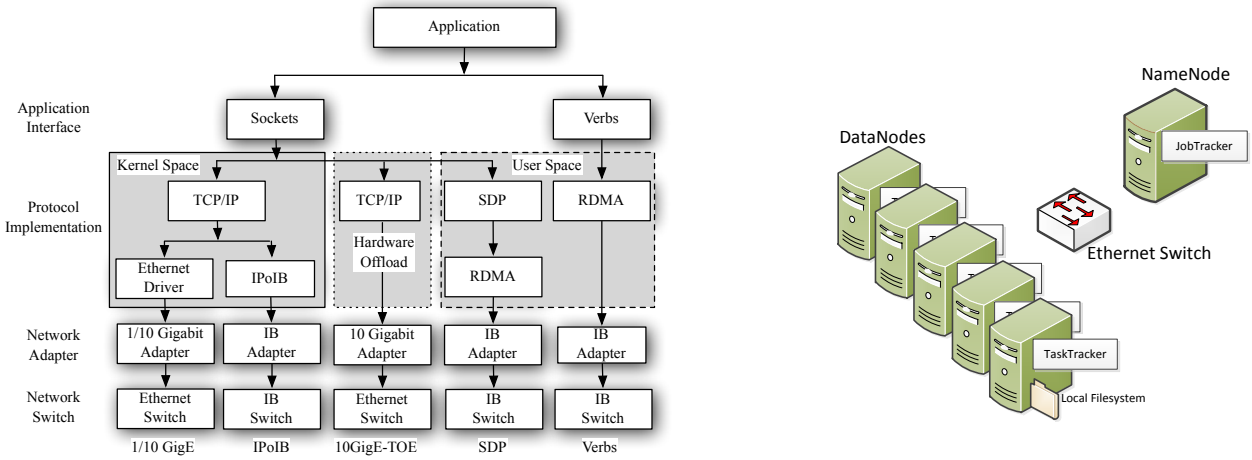
InfiniBand has started making inroads into the commercial domain with the recent convergence around RDMA over Converged Enhanced Ethernet (RoCE) [25].

1) InfiniBand Architecture: The InfiniBand specification clearly demarcates the duties of hardware (such as Host Channel Adapters (HCAs)) and software. The interaction between software and HCAs is carried out by the *verbs* layer, which is described in the following section. The InfiniBand fabric can consist of multi-thousand nodes with multiple adapters. Typically, InfiniBand networks are deployed using the fat-tree topology, which provides constant bisection bandwidth. However, recently, some large deployments have also adopted 3-D torus and hypercube topologies. InfiniBand provides flexible static routing. The routing tables at switches can be configured using the Subnet Manager. For more details on InfiniBand please refer to specification documents available from [11].

2) InfiniBand Verbs Layer: Upper-level software uses an interface called *verbs* to access the functionality provided by HCAs and other network equipment (such as switches). This is illustrated in Figure 1(a) (to the extreme right). Verbs that are used to transfer data are completely OS-bypassed. The verbs interface is a low-level communication interface that follows the Queue Pair (or communication end-points) model. Queue pairs are required to establish a queue pair between themselves. Each queue pair has a certain number of work queue elements. Upper-level software places a work request on the queue pair that is then processed by the HCA. When a work element is completed, it is placed in the completion queue. Upper level software can detect completion by polling the completion queue.

Additionally, there are different types of Queue Pairs based on the type of transport used. There are Reliably Connected (RC) queue pairs that provide reliable transmission (retransmissions after packet losses are performed by the HCA). These RC queue pairs need to be established uniquely for each communicating pair. This implies an $O(n^2)$ memory usage (for a system with N processes). Another type of queue pair is the Unreliable Datagram (UD). This queue pair type does not provide reliable transmission although it has a significant memory advantage – only one UD QP is capable of communicating with all remote processes. Thus, the memory usage of UD QP is $O(n)$ (for a system with N processes).

3) InfiniBand IP Layer: InfiniBand also provides a driver for implementing the IP layer. This exposes the InfiniBand device as just another network interface available from the system with an IP address. Typically, Ethernet interfaces are presented as `eth0, eth1` etc. Similarly, IB devices are presented as `ib0, ib1` and so on. This interface is presented in Figure 1(a) (second from the left, named IPoIB). It does not provide OS-bypass. This layer is often called "IP-over-IB" or IPoIB in short. We will use this terminology in the paper. There are two modes available for IPoIB. One is the datagram mode, implemented



(a) Networking Layers, OS-Bypass and Hardware Offload

(b) HDFS Architecture

Fig. 1. Overview of High-Performance Networking Stack and Hadoop Distributed File System

over Unreliable Datagram (UD), and the other is connected mode, implemented over RC. The connected mode offers better performance since it leverages reliability from the hardware. In this paper, we have used connected mode IPoIB.

4) InfiniBand Sockets Direct Protocol: Sockets Direct Protocol (SDP) [4] is a byte-stream transport protocol that closely mimics TCP socket's stream semantics. It is illustrated in Figure 1(a) (second from the right, named SDP). It is an industry-standard specification that utilizes advanced capabilities provided by the network stacks to achieve high performance without requiring modifications to existing sockets-based applications. SDP is layered on top of IB message-oriented transfer model. The mapping of the byte-stream protocol to the underlying message-oriented semantics was designed to transfer application data by one of two methods: through intermediate private buffers (using buffer copy) or directly between application buffers (zero-copy) using RDMA. Typically, there is a threshold of message size above which zero-copy is used. Using SDP, complete OS-bypass can be achieved. For the sake of simplicity, the figure only shows the RDMA implementation of SDP, as indicated above, for small messages, SDP can use a buffered mode.

B. 10Gigabit Ethernet Overview

In an effort to improve bandwidth in data-center environments, 10Gigabit Ethernet was standardized. It was also realized that to support increasing speeds of network hardware, it may not be completely feasible to only offer traditional sockets interface. This is to reduce memory copies while transferring messages. Towards that effort, iWARP [21] standard was introduced. iWARP is very similar the the verbs layer used by InfiniBand, with the exception of requiring a connection manager. In fact, the OpenFabrics [19] network stack provides an unified interface for both iWARP and InfiniBand. In addition to iWARP, there are also hardware accelerated versions of TCP/IP available. These are called TCP Offload Engines (TOE), which use hardware offload. Figure 1(a) shows this option (in the middle, named 10GigE-TOE). The benefits of TOE are to maintain full socket streaming semantics and implement that efficiently in hardware. We used 10Gigabit Ethernet adapters from Chelsio Communications for this study.

Recently, 10Gigabit Ethernet and InfiniBand are witnessing a convergence. In particular, InfiniBand adapters from Mellanox can be configured to run on Ethernet networks. The software stack by OpenFabrics also provides a unified environment for both networks to support same applications without any code changes.

C. Solid State Drive (SSD) Overview

Conventional hard drives are constrained by the mechanical rotating disk, which results in poor random access performance and excessively high power consumption [8, 9, 16]. Flash memory is a kind of electronic-erasable non-volatile memory. Due to its compact factor and low power consumption, it has been widely adopted in consumer electronics.

However some intrinsic characteristics of flash memory, such as asymmetric read/write time and low write-durability have inhibited its adoption into main stream data storage systems. Recently significant amount of studies

have been carried out to cope with the insufficiency of flash memory [12, 14, 15]. The outcome of these studies is a Flash Translation Layer (FTL) that maps data's logical address to physical address in flash memory. This FTL layer in its essence is similar to a Log-Structured file system [22] in that, it converts all writes, random or sequential, to sequential appending to the end of the log. By doing so it also achieves an even wear-leveling [13] which avoids the premature wear-out in a portion of the flash memory.

With the advancement at both research community and Industry, today flash memory based Solid State Drive (SSD) [1, 3] has emerged as a viable alternative to mechanical disks in main stream computer clusters [20]. These high capacity SSDs have implemented a sophisticated FTL layer that achieves both high performance and reliable data durability. They exhibit many technical advantages over mechanical disks such as outstanding random-access performance, high sequential-access throughput, low power consumption, compact form factor and better shock resistance.

D. Hadoop Distributed Filesystem (HDFS) Overview

HDFS (Hadoop Distributed File System) is a distributed user level filesystem to store and manage the data in a Hadoop cluster. As illustrated in Figure 1(b), HDFS includes a dedicated node called NameNode to store all meta-data and multiple other nodes called DataNode to store applications data. The Ethernet network is used to connect all nodes. HDFS is implemented in Java and provides the portability across heterogeneous hardware and software platforms. Files in HDFS are split into smaller blocks, typically 64MB, and each block is stored as an independent file in the local file system of DataNodes. Through the DataNode abstraction which is independent of local storage, HDFS supports the physical filesystem heterogeneity. Each block of a HDFS file is replicated at multiple DataNodes, typically 3 replicas. Through replicating application data, HDFS provides the data durability. The NameNode manages the namespace tree and the physical location of each file. HDFS client contacts the NameNode to perform file system operations.

When a client application reads a HDFS file, it first requires the NameNode to check its access permission and gets the list of DataNodes hosting replicas of all blocks. Then, it sends the requirements to the "closest" DataNode and requests a specific block. Next, a socket connection is created between the client and the DataNode. The data is transferred to the client application. When a client application writes a HDFS file, it first splits the file into HDFS blocks (64MB) and requires the NameNode to get the list of DataNodes to host replicas of the each block which is handled by simultaneous threads. If the client application is running on a DataNode, the first replica of the file is written into the local file system on the current DataNode. If the client application isn't running on a DataNode, a socket connection is created between the client and the first DataNode. The client splits the block into smaller packets, typically 4KB, and starts a pipeline: the client sends a packet to the first DataNode; the first DataNode receives this packet, writes it to the local file system, and flushes it to the next DataNode. A DataNode can receive the data from a previous node and at the same time forward the data to the next node. When all nodes in this pipeline write the block into local filesystem successfully, the block write is finished and then DataNodes update the block physical information to the NameNode.

III. Benefits of Modern Interconnects on Cloud Computing Middleware

High-Performance interconnects provide low latency (1-2μs) and high bandwidth (up to 32Gbps). In addition, they can support this high rate and low latency with very little CPU utilization. Typical values of CPU utilization of 2%-5% are commonly reported while transferring messages at wire speed on commodity machines running mainline Linux kernels.

The HPC community has very effectively leveraged the efficiency of these interconnects. The Message Passing Library (MPI) is a standard for parallel programming and is very widely used by many scientific applications. It provides a message-boundary oriented interface to send, receive messages along with remote-memory access (RMA). MVAPICH and MVAPICH2 [17] are high-performance implementations of the MPI standard on InfiniBand. MVAPICH/MVAPICH2 are implemented directly on top of InfiniBand verbs and provide the highest performance.

Figure 2 presents the efficiency of the Top500 systems. The data is collected from top500.org. The efficiency metric is calculated as the ratio of peak flops (floating point operations) of the hardware to sustained performance reported by the Linpack benchmark. We observe from this figure that systems connected with InfiniBand provide very good efficiencies, often in the 90% to 95% range (with proper tuning). In fact, efficiencies offered by InfiniBand and 10Gigabit Ethernet rival those of proprietary custom designed interconnection networks such as those from IBM Blue-Gene and Cray. 1Gigabit Ethernet systems using MPI libraries over traditional sockets can only provide about 50% efficiency. It is to be noted that the IB-GPU/Cell systems have lower efficiency, but that is due to the inability to achieve peak flops on NVIDIA GPUs or IBM Cell

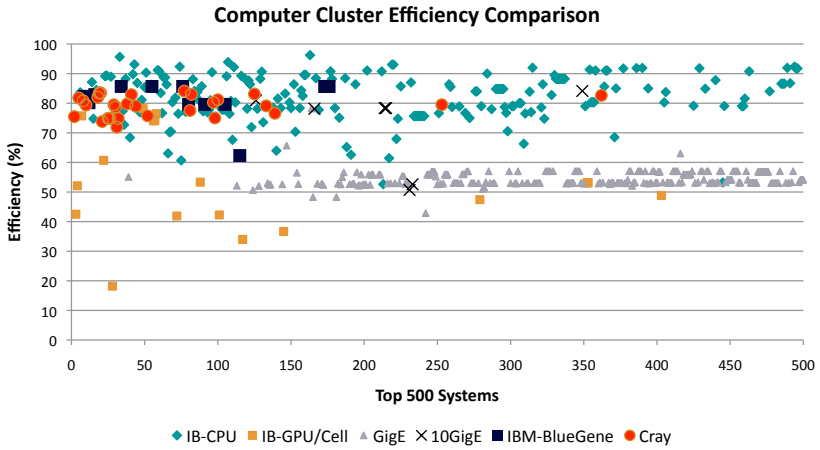


Fig. 2. Efficiency data from Top500 Compute Systems Ranking November 2010

blades. This is an issue with peak advertised flops using special instructions useful for graphics, but not exploited by Linpack benchmark. This is definitely not a limitation of IB as such, rather a CPU vs. GPU issue.

We believe that using high-performance interconnects, even data-centers can boost their efficiency significantly, as have the Top500 compute systems.

IV. Experimental Evaluation and Results

In this section, we conduct several popular Hadoop Benchmarks to evaluate the performance of HDFS with different networking technologies(1GigE, IPoIB, SDP, and 10GigE) using either hard drive or SSD as storage devices.

A. Experiment Configurations

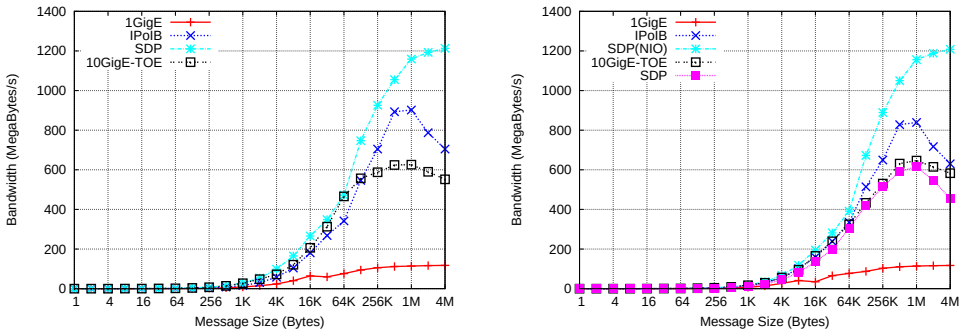
An InfiniBand Linux cluster is used in the evaluation. Each node in the cluster has eight processor cores on two Intel Xeon 2.33GHz Quad-core CPUs, 6 GB Main memory, and a 250GB ST3250310NS Hard drive. Each node is equipped with a Mellanox MT25208 DDR (16Gbps) HCA and 10GigE adapter by Chelsio Communications (T320). HDFS is implemented over sockets. In this paper, we focus on understanding the performance characteristics of HDFS when the sockets layer itself is configured to run on InfiniBand and 10Gigabit Ethernet. We choose three different modes to run sockets. The combinations are shows in Figure 1(a) and marked with the same names used in the rest of this section. IPoIB, SDP, 10GigE-TOE and 1GigE. The 1Gigabit Ethernet is the on-board Ethernet device.

All nodes run RHEL 5.3 and Hadoop framework 0.20.2 with Sun Java SDK 1.6.0. In our experiment, one node is dedicated as the NameNode server, and another node as JobTracker. We vary the number of DataNode to be 2/4/8. Each DataNode also works as a TaskTracker. The Hadoop replication factor is set to 3 for all experiments. This is the recommended number of replicas. Out of the 8 DataNodes, four nodes have an Intel X-25E 64GB SSD as an alternative storage device.

B. Microbenchmark Level Evaluation

HDFS is written in Java for portability reasons. It uses sockets programming API in Java for network communication. In order to understand Java socket communication performance on different networks and protocols, we designed and implemented a micro benchmark to test point-to-point bandwidth. The test has two processes - client and server. The client sends a particular size message for a certain number of iterations. When the server has received all the messages, it sends an acknowledgment back to the client. We have implemented this benchmark in both C and Java.

Figure 3(a) illustrates bandwidth achieved by the C version of the benchmark. The InfiniBand network is 16Gbps (DDR - Double Data Rate), therefore bandwidth achieved is greater than 10Gigabit Ethernet. Figure 3(b) illustrates bandwidth with the corresponding Java version of the benchmark. We have developed two versions of the Java benchmark. One uses normal Java arrays (`byte[]`) for the messages exchanged between server and client. The other version uses Java NIO direct allocated buffers for the messages. As can be seen from the figure, the NIO version can achieve very good bandwidth, similar to the C version. By doing this micro-benchmark level evaluation, we learn that provided there is some care taken in the messaging of



(a) Bandwidth with C

(b) Bandwidth with Java

Fig. 3. Bandwidth comparison between C and Java on various network interfaces and technologies

HDFS design, it has the potential to achieve speeds that are possible in a lower-level language, such as C. We also note that while HDFS uses NIO buffers, it does not use the `allocateDirect` method for getting message buffers at DataNodes. Without using direct buffers, the JVM may not use native methods for I/O. Also, direct buffers are allocated outside the garbage collection heap. However, allocation of direct buffers may be more expensive. Therefore, HDFS needs to be redesigned keeping these factors in consideration.

C. Sequential Write/Read Performance of DFS I/O Benchmark

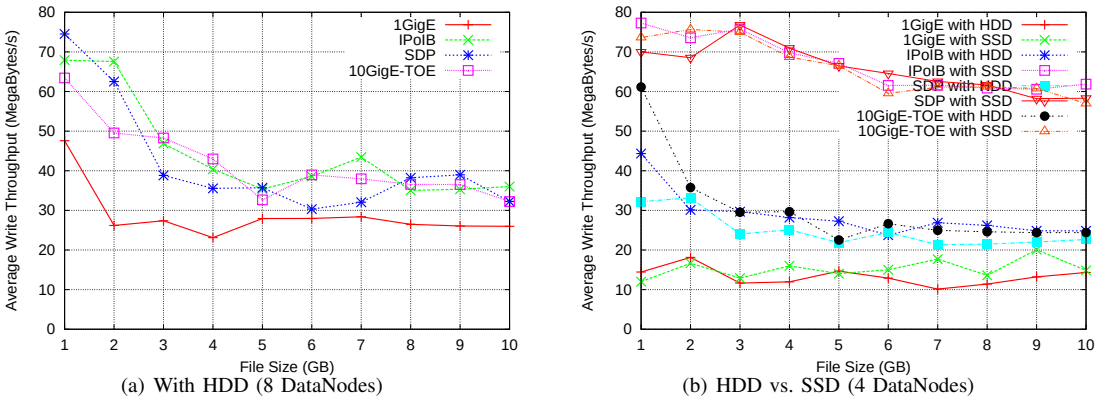
DFSIO is a file system benchmark as part of the Hadoop distribution that measures the I/O performance of HDFS. It is implemented as a Hadoop job in which each map task opens one file to perform sequential write or read, and measures the data IO size and execution time of that task. There is a single reduce task followed by a post-processing task, which aggregates the performance results of all the map tasks [10]. In our experiments we start 2 map tasks each writing/reading a file to/from 8 DataNode servers, with size of each file ranging from 1GB to 10GB. Due to space constraints, only the results with two map tasks and two data files are shown, but the same trend is observed in all tests using varied number of map tasks.

Figure 4(a) shows the averaged sequential write throughput for each map task using 1GigE, IPoIB, SDP and 10GigE (Chelsio), respectively. As indicated in Figure 4(a), better network bandwidth helps improve the write performance. Compared to the baseline 1GigE, IPoIB/SDP/10GigE can achieve significant performance gains. For smaller files of 2GB, we see a bigger improvement greater than 100%. For larger files, the improvement is still about 35%.

Shafer et. al have described HDFS limitations when it comes to I/O scheduling in [23]. This work shows that the current HDFS implementation is not designed to use a pool of I/O threads. Some of these limitations are alleviated by SSDs. Although, currently, SSDs have not reached the capacities per dollar of HDDs, they are gradually being adopted for staging and for use with medium sized datasets. In this experiment, we used SSDs to see if the performance of HDFS can be improved with SSDs. Most notably, we wanted to observe if SSDs result in a higher impact of high-performance networks (especially since the I/O bottlenecks are alleviated due to a different storage technology). Due to limited amount of hardware resources, we used 4 DataNodes each with an Intel X-25E 64GB SSD to store the data files, and rerun the above tests with 2 map tasks. Figure 4(b) compares the average write throughput for each map task using Hard Drive and SSD with varied interconnections and protocol stacks.

With 1GigE, SSD improves the write performance by up to 53% over hard drive. Much higher improvements are obtained when advanced interconnections and protocol stacks come into play. In the case of 1GigE, the system tends to be network bandwidth bounded, so SSD's advantages are not clearly exhibited. As the network bandwidth is increased, the system becomes more and more storage bandwidth bounded. Therefore the excellent throughput of SSD is manifested more prominently. SSD achieves 2.55 times the throughput of HDD when IPoIB is used. The throughput with SSD is up to 3.19/2.54 times of HDD when using SDP/10GigE respectively.

In the sequential read test, each DataNode reads data locally, verifies the correctness of each data chunk and discards it without transferring data over the network. This results only reflect the local I/O throughput on each



(a) With HDD (8 DataNodes) (b) HDD vs. SSD (4 DataNodes)

Fig. 4. DFS-IO Sequential Write Throughput (Higher is Better)

DataNode. We feel this is orthogonal to our intentions in this paper which focus on how network interconnects can affect HDFS performance. Therefore they are not included in this paper.

D. RandomWriter Benchmark Evaluation

RandomWriter is a built-in benchmark inside Hadoop distribution. It runs a MapReduce job with N maps per node. Each map generates 1GB of random binary data containing varied-sized key-value pairs, and write these pairs to the HDFS sequence file. The reduce phase is not used [28]. We let each DataNode host 5 maps, with total amount of data to be 10/20/40 GB on 2/4/8 DataNodes. Figure 5(a) shows the execution time of RandomWriter using 1GigE, IPoIB, SDP and 10GigE(Chelsio), respectively. With 2 DataNodes, HDFS implicitly adjusts the data replication factor to be two instead of the default three replicas, resulting in less cost incurred to transfer the duplication over the network. Therefore the execution time is much less than with 4/8 DataNodes. As more data is transferred over the network with 4/8 DataNodes, benefits of advanced interconnection technologies and protocols can be observed more obviously. At 4 DataNodes, IPoIB/SDP/10GigE drives down the execution time by 15%/22%/25%, respectively. Larger improvements of 29%/29%/30% are obtained when more storage bandwidth is provisioned using 8 DataNodes.

We then substituted disk drives on 4 DataNodes with SSDs, and rerun the RandomWriter benchmark to measure the performance improvement caused by SSD. As depicted in Figure 5(b), SSD helps reduce the execution time by 50% with 1GigE on 2 DataNodes. With 4 DataNodes and SDP, SSD cuts down the execution time by 59% over disk drives.

E. Sort Benchmark Evaluation

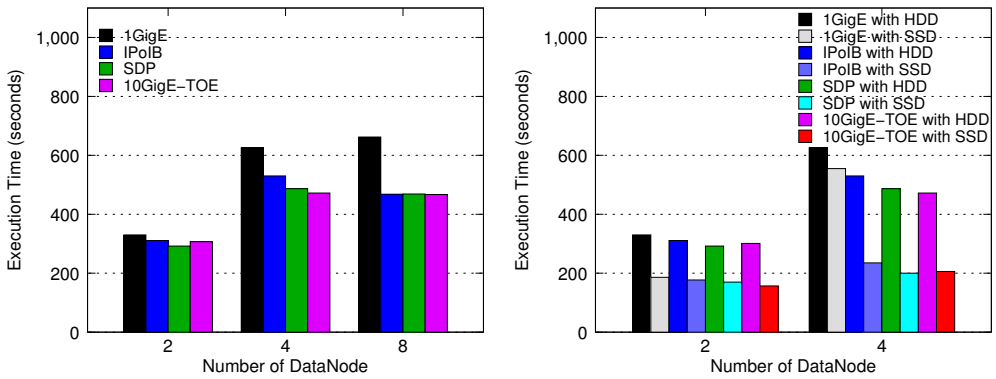
Sort is often used as a baseline benchmark for HDFS. The sorting program has been pervasively accepted as an important performance indicator of MapReduce, because sorting is an intrinsic behavior of the MapReduce framework. In our experiments, the input data of Sort is generated using the RandomWriter. At the beginning of its execution, Sort maps the tasks to all data nodes. Each task then reads local data and performs sorting in parallel. A reduce phase follows the sorting phase to merge the locally-sorted data chunks from all data nodes and writes them to a new file. Figure 6(a) shows the execution time of Sort on 2/4/8 DataNodes.

Since Sort is primarily bounded by the disk IO bandwidth on each DataNode to load data at sorting phase, advanced network technologies and protocols can only help in the reduce phase to transfer partially-sorted data. Therefore we observe a slight improvement of up to 11% over the baseline 1GigE.

Given that Sort is more likely to be disk IO bandwidth bounded, we replaced the disk drives on 4 DataNodes with SSDs to evaluate the benefit of high performance storage devices. As expected, in Figure 6(b), SSD helps shrink the execution time by 28% over the disk drive when using 2 DataNodes with 1 GigE. If SDP is used as the network communication mechanism, SSD shortens the execution time by 48% over disk drive on 4 DataNodes.

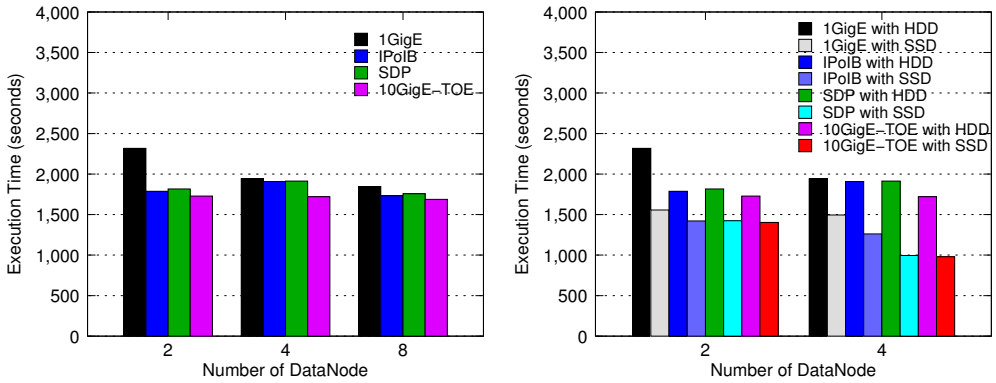
V. Related Work

The MapReduce programming model was introduced recently in 2004 by Dean et. al in [7]. Doug Cutting et al. have implemented an Open-source version of MapReduce – Hadoop, based on the Hadoop Distributed File System (HDFS) [27]. Apache Hadoop is designed to work on



(a) With HDD (b) HDD vs. SSD

Fig. 5. Execution time of Random Write (Lower is Better)



(a) With HDD (b) HDD vs. SSD

Fig. 6. Execution time of Sort (Lower is Better)

a wide variety of hardware. It is written in Java. Recently, [23], Shafer et. al. have analyzed and revealed several HDFS performance bottlenecks: the first one is the architecture bottleneck in Hadoop implementation, which delays new MapReduce tasks scheduled to DataNodes; the second one is portability bottleneck: in order to maintain portability across heterogeneous platforms, HDFS ignores the optimization on the native storage resource and leads to local disk access becoming a bottleneck. In [24], Shvachko et. al. from Yahoo! have described HDFS detailed design and implementation. They reveal the bandwidth and not latency is more important to HDFS performance. They have also discussed HDFS performance with the aggregated read and write bandwidth on DataNodes and the throughput on NameNode in their cluster. In [26], Tantisiriroj et. al. have compared PVFS and HDFS, and illustrated PVFS can provide the comparable performance to HDFS in the Internet services environment. Through real applications such as sort, they also reveal HDFS is better when it comes to support MapReduce, due to its design characteristics such as move code to data to improve locality, process data sequentially, and avoid data random access patterns. In the recent past, there has been some work towards the applicability of high-performance interconnects in data-center environment [5, 18, 30, 31]. This work mainly focused on developing low-level substrates to expose RDMA capabilities of InfiniBand and 10Gigabit iWARP Ethernet. This work, however, was done before the Hadoop project became popular and there was an ecosystem of open-source data-center middleware. The aim of this paper is to attempt to bring the benefits of modern interconnects and protocols to data-center middleware by the way of Hadoop open-source software.

VI. Conclusions and Future Work

In this paper we have attempted to answer the question of whether high-performance interconnects and protocols can have a positive impact on Hadoop Distributed Filesystem (HDFS) performance. In order to see the impact

of faster interconnection network, we examined various network options to enable accelerated sockets. In the case of InfiniBand, we explored the use of IP over IB and Sockets Direct Protocol on Mellanox adapters. In the case of 10Gigabit Ethernet, we explored the use of a TCP Offload Engine (TOE) available from Chelsio. We also investigated the performance of HDFS workloads with Solid State Drives. Our findings reveal that a faster interconnect and protocols can improve the performance of HDFS significantly. The impact is even more pronounced when SSDs are used for storage.

We observe up to 11%, 30% and 100% performance improvement for the sort, random write and sequential write benchmarks using magnetic disks (HDD). We also find that with the emerging trend of Solid State Drives, having a faster interconnect makes a larger impact as local I/O costs are reduced. We observe up to 48%, 59% and 219% improvement for the same benchmarks when SSD is used in combination with advanced interconnection networks and protocols. We also discuss opportunities in HDFS implementation to improve the chances of performance increase with better interconnects. For example, the use of direct allocated buffers on DataNodes will enable Sockets Direct Protocol on InfiniBand to use zero-copy protocols and boost bandwidth.

We will continue working in this area in the future. We are currently investigating strategies to reduce HDFS communication overheads. We are are also looking into HBase and its use of HDFS for implementing random access to large amounts of data in real time.

VII. Acknowledgments

This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342, #CCF-0702675, #CCF-0833169, #CCF-0916302 and #OCI-0926691.

We would like to thank Dr. Chet Murthy and Dr. Matthew Arnold of IBM T. J. Watson Research Center for many helpful discussions. We would also like to thank Dror Goldenberg, Amir Vadai and Nir Kriss of Mellanox Technologies for their help in getting Java and SDP to work together.

References

[1] Fusion-IO SSD. http://www.fusionio.com/.

[2] InfiniBand Trade Association. http://www.infinibandta.com.

[3] Intel SSD. http://www.intel.com/design/flash/nand/.

[4] Sockets Direct Protocol. `http://www.infinibandta.com`.

[5] P. Balaji, H. V. Shah, and D. K. Panda. Sockets vs RDMA Interface over 10-Gigabit Networks: An In-depth analysis of the Memory Traffic Bottleneck. In *Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT), in conjunction with IEEE Cluster*, 2004.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *OSDI'06: Seventh Symposium on Operating System Design and Implementation*. USENIX Association, 2006.

[7] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *In OSDI04: Proceedings of the 6th conference on Symposium on Opearting Systems Design and Implementation*. USENIX Association, 2004.

[8] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang. DiskSeen: Exploiting Disk Layout and Access History to Enhance I/O Prefetch. In *Proceedings of USENIX07*, 2007.

[9] C. Gniady, Y. C. Hu, and Y.-H. Lu. Program Counter Based Techniques for Dynamic Power Management. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, HPCA '04, Washington, DC, USA, 2004. IEEE Computer Society.

[10] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *ICDE Workshops*, pages 41–51, 2010.

[11] Infiniband Trade Association. http://www.infinibandta.org.

[12] Jesung Kim and Jong Min Kim and Noh, S.H. and Sang Lyul Min and Yookun Cho. A space-efficient flash translation layer for CompactFlash systems . *Consumer Electronics, IEEE Transactions on*, 48(2):366–375, May 2002.

[13] Kawaguchi, Atsuo and Nishioka, Shingo and Motoda, Hiroshi. A flash-memory based file system. In *Proceedings of the USENIX 1995 Technical Conference Proceedings on USENIX 1995 Technical Conference Proceedings*, pages 13–13, 1995.

[14] Lee, Sang-Won and Park, Dong-Joo and Chung, Tae-Sun and Lee, Dong-Ho and Park, Sangwon and Song, Ha-Joo. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.*, 6(3):18, 2007.

[15] Lee, Sungjin and Shin, Dongkun and Kim, Young-Jin and Kim, Jihong. LAST: locality-aware sector translation for NAND flash memory-based storage systems. *SIGOPS Oper. Syst. Rev.*, 42(6):36–42, 2008.

[16] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou. C-Miner: Mining Block Correlations in Storage Systems. In *Proceedings of the 3rd USENIX Symposium on File and Storage Technologies (FAST 04)*, pages 173–186, 2004.

[17] MVAPICH2: High Performance MPI over InfiniBand, 10GigE/iWARP and RoCE. http://mvapich.cse.ohio-state.edu/.

[18] S. Narravula, A. Mamidala, A. Vishnu, K. Vaidyanathan, and D. K. Panda. High Performance Distributed Lock Management Services using Network-based Remote Atomic Operations. In *Int'l Symposium on Cluster Computing and the Grid (CCGrid)*, 2007.

[19] OpenFabrics Alliance. http://www.openfabrics.org/.

[20] X. Ouyang, S. Marcarelli, and D. K. Panda. Enhancing Checkpoint Performance with Staging IO and SSD. *Storage Network Architecture and Parallel I/Os, IEEE International Workshop on*, pages 13–20, 2010.

[21] RDMA Consortium. Architectural Specifications for RDMA over TCP/IP. http://www.rdmaconsortium.org/.

[22] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10, 1992.

[23] J. Shafer, S. Rixner, and A. L. Cox. The Hadoop Distributed Filesystem: Balancing Portability and Performance. In *Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'10)*, pages 122–133, 2010.

[24] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Proceedings of the 26th IEEE International Symposium on Mass Storage Systems and Technologies (MSST'10)*, pages 1–10, 2010.

[25] H. Subramoni, P. Lai, M. Luo, and D. K. Panda. RDMA over Ethernet - A Preliminary Study. In *Proceedings of the 2009 Workshop on High Performance Interconnects for Distributed Computing (HPIDC'09)*, 2009.

[26] W. Tantisiriroj, S. Patil, and G. Gibson. Data-intensive file systems for Internet services: A rose by any other name ... Technical report, Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-08-114, 2008.

[27] The Apache Software Foundation. The Apache Hadoop Project. http://hadoop.apache.org/.

[28] Tom White. *Hadoop: The Definitive Guide*. O'Reilly.

[29] Top500. Top500 Supercomputing systems, November 2010. http://www.top500.org.

[30] K. Vaidyanathan, S. Narravula, P. Balaji, and D. K. Panda. Designing Efficient Systems Services and Primitives for Next-Generation Data-Centers. In *Workshop on NSF Next Generation Software(NGS) Program; held in conjunction with IPDPS*, 2007.

[31] K. Vaidyanathan, S. Narravula, P. Lai, and D. K. Panda. Optimized Distributed Data Sharing Substrate in Multi-Core Commodity Clusters: A Comprehensive Study with Applications. In *Int'l Symposium on Cluster Computing and the Grid (CCGrid)*, 2008.