High Performance RDMA-based Design of HDFS over InfiniBand*

N. S. Islam[1], M. W. Rahman[1], J. Jose[1], R. Rajachandrasekar[1], H. Wang[1], H. Subramoni[1], C. Murthy[2], and D. K. Panda[1]

[1] *Department of Computer Science and Engineering, The Ohio State University*
{islamn, rahmanmd, jose, rajachan, wangh, subramon, panda}@cse.ohio-state.edu

[2] *IBM T.J Watson Research Center Yorktown Heights, NY*
{chet}@watson.ibm.com

***Abstract*—Hadoop Distributed File System (HDFS) acts as the primary storage of Hadoop and has been adopted by reputed organizations (Facebook, Yahoo! etc.) due to its portability and fault-tolerance. The existing implementation of HDFS uses Java-socket interface for communication which delivers suboptimal performance in terms of latency and throughput. For data-intensive applications, network performance becomes key component as the amount of data being stored and replicated to HDFS increases. In this paper, we present a novel design of HDFS using Remote Direct Memory Access (RDMA) over InfiniBand via JNI interfaces. Experimental results show that, for 5GB HDFS file writes, the new design reduces the communication time by 87% and 30% over 1Gigabit Ethernet (1GigE) and IP-over-InfiniBand (IPoIB), respectively, on QDR platform (32Gbps). For HBase, the Put operation performance is improved by 26% with our design. To the best of our knowledge, this is the first design of HDFS over InfiniBand networks.**

I. Introduction

The 2011 International Data Corporation (IDC) study [1] on Digital Universe indicated the beginning of 'Information Age', where the foundation of economic value is largely derived from information vs. physical things. The rate of information growth appears to be exceeding Moore's Law and it is forecasted that 35 zettabytes of data will be generated and consumed by the end of this decade. IT organizations everywhere appear to be bearing the brunt of this transition. Many organizations employ MapReduce programming model which has emerged as a scalable way to perform data-intensive computations. Hadoop is a popular open-source implementation of MapReduce programming model and Hadoop Distributed File System (HDFS) is the underlying file system of Hadoop.

With the advent of Information Age, Cloud Computing systems are being widely deployed on High Performance Computing (HPC) Clusters [2]. A recent example is the 'Greenplum Analytics Workbench' [3], which is a 1000+ node HPC cluster, for running regular integration testing on the Apache Hadoop trunk. Even though such systems are being used for Hadoop deployments, current Hadoop middleware components do not leverage many HPC cluster features, especially the high performance communication features. High performance networks such as InfiniBand [4] provide low latency and high throughput data transmission. Over the past decade, scientific and parallel computing domains, with the Message Passing Interface (MPI) as the underlying basis for most applications, have made extensive usage of these advanced networks. Implementations of MPI, such as MVAPICH2 [5], achieve low one-way latencies in the range of 1-2μs. On the other hand, even the best implementation of sockets on InfiniBand achieves 20-25μs one-way latency [6]. Recent research works [6–8] throw light to the huge performance improvements possible for different cloud computing middlewares using InfiniBand networks and faster disk technologies such as SSDs. HDFS is a communication intensive middleware because of its distributed nature. All existing communication protocols [9] of HDFS are layered on top of TCP/IP. Due to the byte stream communication nature of TCP/IP, multiple data copies are required, which results in poor performance in terms of both latency and throughput. Consequently, even though the underlying system is equipped with high performance interconnects such as InfiniBand, HDFS cannot fully utilize the hardware capability and obtain peak performance. These issues lead us to the following broad questions and design challenges:

1) How does the network performance impact the overall HDFS performance?
2) How can we re-design HDFS to take advantage of high performance interconnects such as InfiniBand and exploit advanced features such as RDMA?
3) What will be the performance improvement of HDFS operations with the new RDMA-based design over InfiniBand?
4) Can we observe the performance improvement for other cloud computing middlewares such as HBase, which use HDFS as the underlying filesystem?

In this work, we address these research challenges. We perform detailed profiling and analysis to identify the impact of network performance on overall HDFS performance for different network interconnects. We present a hybrid HDFS design that incorporates RDMA over InfiniBand, while also supporting conventional sockets interface. We use the in-house InfiniBand light weight communication library, Unified Communication Runtime (UCR) [10], in our design. To the best of our knowledge, this is the first design of HDFS over InfiniBand networks.

We also perform detailed evaluation of the RDMA-capable HDFS design and compare the performance with that of 1 GigE, IPoIB and 10 GigE networks. We perform detailed tuning of HDFS packet-size for different networks and identify

*This research is supported in part by U.S. Department of Energy grant #DE-FC02-06ER25755; National Science Foundation grants #CCF-0916302 and #CCF-0937842; and a grant from IBM Open Collaborative Research.

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

the optimal packet size for each interconnect. Our evaluation shows that, the new design reduces the communication time by 87% over 1 GigE and 30% over IPoIB for 5 GB HDFS file writes. Moreover, we observe that for HBase (Hadoop Database), which uses HDFS as the underlying filesystem, the performance of *Put* operation (1 KB size) with multiple servers is improved by 26% with our new design.

The rest of the paper is organized as follows. In Section II we present background about the key components involved in our design. Section III provides an overview of the socket-based communication in the existing HDFS. In Section IV we propose the detailed design of HDFS over RDMA. In Section V, we describe our experiments and evaluation and Section VI illustrates the performance benefits of HBase using this new design. Related works are discussed in Section VII, and in Section VIII we present conclusions and future works.

II. Background

A. Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is a distributed file system which is used as the primary storage for Hadoop cluster. Figure 1(a) illustrates the basic architecture of HDFS. An HDFS cluster consists of two types of nodes: NameNode and DataNode. The NameNode manages the file system namespace. It maintains the file system tree and stores all the meta data. The DataNodes on the other hand, act as the storage system for the HDFS files. HDFS divides large files into blocks of size 64 MB. Each block is stored as an independent file in the local file system of the DataNodes. HDFS usually replicates each block to three (default replication factor) DataNodes. In this way, HDFS guarantees data availability and fault-tolerance.

The HDFS client contacts the NameNode during any kind of file system operations. When the client wants to write a file to HDFS, it gets the block IDs and a list of DataNodes for each block from the NameNode. Each block is split into smaller packets and sent to the first DataNode in the pipeline. The first DataNode then replicates each of the packets to the subsequent DataNodes. Packet transmission in HDFS is pipelined; a DataNode can receive packets from the previous DataNode while it is replicating data to the next DataNode. If the client is running on a DataNode, then the block will be first written to the local file system. When a client reads an HDFS file, it first contacts the NameNode to check its access permission and gets the block IDs and locations for each of the blocks. For each block belonging to the file, the client connects with the nearest DataNode and reads the block.

B. InfiniBand Overview

InfiniBand [4] is an industry standard switched fabric that is designed for interconnecting nodes in High End Computing (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 [11] rankings in June 2012 reveal that more than 41% of the computing systems use InfiniBand as their primary interconnect. 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) [12].

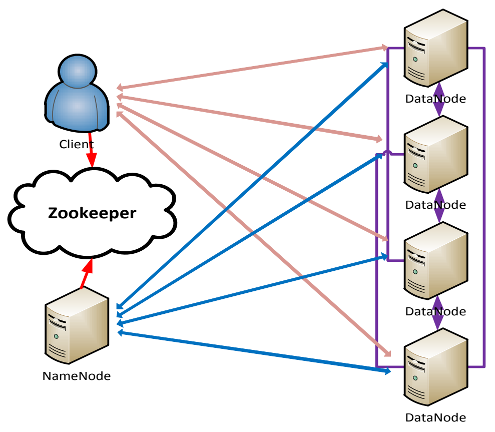
1) InfiniBand Verbs: InfiniBand Host Channel Adapters (HCA) and other network equipments can be accessed by the upper layer software using an interface called *Verbs*. This is illustrated in Figure 1(b) (to the extreme right). The verbs interface is a low level communication interface that follows the Queue Pair (or communication end-points) model. Queue pairs are required to establish a channel between the two communicating entities. 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. Verbs that are used to transfer data are completely OS-bypassed.

2) InfiniBand IP Layer: InfiniBand also provides a driver for implementing the IP layer, allowing socket applications to make use of InfiniBand networks. This exposes the InfiniBand device as just another network interface available from the system with an IP address. InfiniBand devices are presented as ib0, ib1 and so on. This interface is presented in Figure 1(b) (second from the left, named IPoIB). However, 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 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.

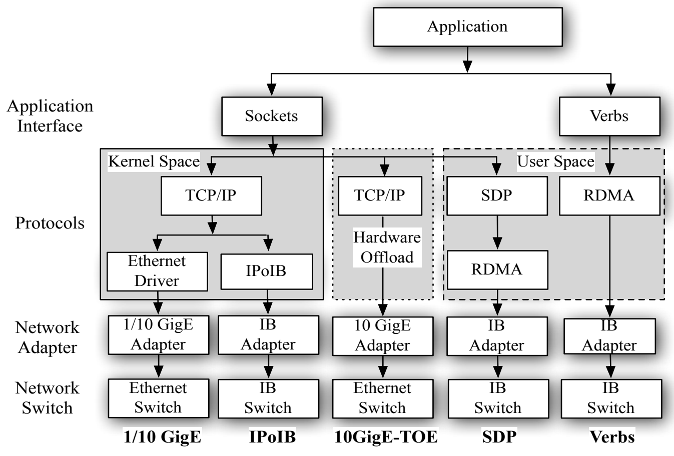
3) InfiniBand Sockets Direct Protocol: Sockets Direct Protocol (SDP) is a byte-stream transport protocol that closely mimics TCP sockets stream semantics. It is illustrated in Figure 1(b) (second from the right, named SDP). It is an industry-standard specification that utilizes advanced capabilities provided by the network stacks to achieve high performance without requiring modifications to existing socket based applications. SDP is layered on top of IB message-oriented transfer model. SDP uses buffered mode for smaller message sizes and direct RDMA for larger message sizes. While SDP offers better performance than IPoIB due to its capability to leverage RDMA, it still does not match the performance of verbs. SDP is no longer supported in OpenFabrics Enterprise Distribution (OFED) [13] stack. So we have not considered SDP for performance evaluations in our study.

C. 10 Gigabit Ethernet Overview

The 10Gigabit Ethernet interconnect was standardized in 2002 with hopes of it being the next big commodity network. The motivation behind it was to have a high-speed network



(a) High-level overview of HDFS architecture



(b) Networking layers, OS-bypass and hardware offload, courtesy [7]

Fig. 1. Overview of HDFS and high-performance networking stacks

which can handle packet losses and drops with retransmissions. However, with high wire-speeds, the overheads imposed by the Ethernet protocol had to be circumvented, by means other than just relying on a fast CPU as in the case of 1Gigabit Ethernet. This led the RDMA consortium to develop the iWARP standard that eliminated the overheads from TCP/IP processing, intermediate buffer copies and application context-switches.

The OpenFabrics [13] network stack provides a 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(b) 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.

D. Solid State Drive (SSD) Overview

Solid-State Disks have amassed a lot of attention over the recent-past owing to significant data-throughput and efficiency gains over traditional spinning-disks. Although it is a mere physical array of fast flash-memory packages, the core-intelligence of an SSD can be attributed to its Flash Translation Layer (FTL) which plays a vital role in the adoption of this technology. Some of major functionality of an SSD, such as Wear leveling, Garbage collection and Logical Block Mapping, is packed into the FTL.

SSDs are efficient with sequential-write workloads, which makes it an ideal candidate for use in the HDFS framework. The Write-Once-Read-Many model adopted by Hadoop also fits well with SSD semantics where overlapping reads and writes degrade the throughput owing to overheads from internal operations, such as cleaning and asynchronous write-back of dirty data from the disk-cache, triggered by a write operation [14]. Our prior work [7] revealed that the use of SSDs in the HDFS DataNode reduces the cost of local I/O, and makes the overheads involved in data-transmission over the network prominent. Based on this conclusion from our earlier work, we use SSDs in our experiments to lessen the I/O bottlenecks when studying the effects of communication over different interconnects.

E. Unified Communication Runtime (UCR)

The Unified Communication Runtime (UCR) [10] is a light-weight, high performance communication runtime, designed and developed at The Ohio State University. It aims to unify the communication runtime requirements of scientific parallel programming models, such as MPI and Partitioned Global Address Space (PGAS) along with those of data-center middleware, such as Memcached [15], HBase [16], MapReduce [17], etc.

UCR is designed as a native library to extract high performance from advanced network technologies. The UCR project draws from the design of MVAPICH and MVAPICH2 software. MVAPICH and MVAPICH2 [5] are popular implementations of the MPI-1 and MPI-2 specifications. They are being used by more than 1,930 organizations in 68 countries and also distributed by popular InfiniBand software stacks and Linux distributions like RedHat and SUSE.

F. Hadoop Database (HBase)

HBase is a Java-based database that runs on top of the Hadoop framework [18]. It is used to host very large tables with many billions of entries and provides capabilities similar to Google's BigTable [19]. It is developed as part of the Apache Software Foundation's Apache Hadoop project [9] and runs on top of HDFS. It flushes its in-memory data to HDFS whenever the size of its memory store reaches a particular threshold (default is 64 MB).

HBase features compression, in-memory operations, and Bloom filters [20] on a per-column basis. Tables in HBase can serve as the input and output for MapReduce [21] jobs run in Hadoop, and may be accessed not only through the Java API but also through REST [22], Avro [23] or Thrift [24] gateway APIs.

Although HBase is not a direct replacement for the classic relational SQL Database, it is gradually becoming essential for many data-driven websites including Facebook's Messaging Platform. The scalability and platform independence of HBase makes it attractive in a data intensive cloud computing environment.

G. Yahoo! Cloud Serving Benchmark (YCSB)

The goal of Yahoo! Cloud Serving Benchmark (YCSB) [25] is to facilitate performance comparisons of different key/value-pair and cloud data serving systems. It defines a core set of benchmarks for four widely used systems: HBase, Cassandra [26], PNUTS [27] and a simple shared MySQL implementation. The core workload of HBase consists of six different workloads and each represents a different application scenario. Zipfian and Uniform distribution modes are used in YCSB for record selection in database. Besides that, customization of workloads is also possible. In addition to these different workloads, there are a number of runtime parameters that can be defined while running YCSB.

III. Overview of Socket-based Communication in HDFS

In this section, we present an overview of the socket-based communication in the existing HDFS. In this paper, we primarily focus on the *HDFS Write* operation. Since *HDFS Write* involves replication also, we believe, write operation is more network-intensive compared to *HDFS Read*. Moreover, in MapReduce programming model, most input data is read locally and consumes no network bandwidth [28]. Figure 2 illustrates the socket-based communications that take place during *HDFS Write* in the existing design.

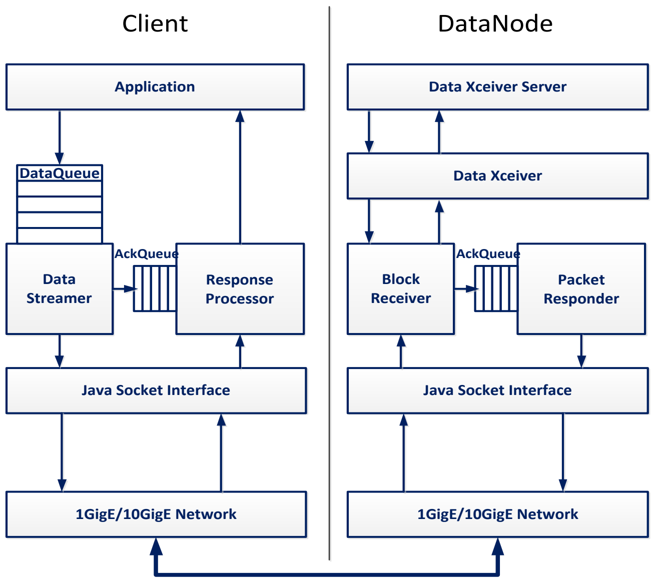


Fig. 2. Socket-based communication in HDFS

The main components involved in the socket-based communication are:

DataStreamer: The DataStreamer thread runs as a daemon in the DFSClient side. Data packets from the upper layer are put into the DataQueue associated with this thread. DataStreamer is responsible for sending the block header and the data packets for each block to different DataNodes via Java socket. Due to the byte-stream model of Java socket, each packet is converted to a stream of bytes before being written to the socket. After sending each packet, it is inserted into the AckQueue. Application performance depends on the speed at which the DataStreamer sends the data packets, as new packets can only be inserted into the DataQueue up to a certain limit.

ResponseProcessor: The ResponseProcessor thread waits for the acknowledgment for each packet. Once the acknowledgment for a packet arrives, the packet is removed from the AckQueue.

DataXceiverServer: In the DataNode, a listener thread is running on DataXceiverServer. This thread keeps on monitoring the Java socket for incoming connection requests.

DataXceiver: When the listener in the DataXceiverServer accepts a socket connection, it creates a DataXceiver thread. The DataXceiver acts as a daemon to process the incoming data. The DFSClient sends write request for a block to the DataXceiver which then parses the header and takes action accordingly (calls writeBlock()). Inside the writeBlock() function, header processing as well as replication takes place and an acknowledgment is sent back to the client. Then a BlockReceiver is created for the incoming block, which waits for receiving packets belonging to that block. The DFSClient, after getting an acknowledgment for the block header, starts sending packets for the corresponding block. The BlockReceiver, upon receiving the byte stream from the socket, deserializes and processes the packet. It is then replicated to the next DataNode in the pipeline and flushed into the local file system. Then the sequence number of the packet is enqueued into the acknowledgment queue. The PacketResponder sends acknowledgment for each packet to either the client or the previous DataNode in the pipeline. Acknowledgments are also sent using Java socket.

IV. Proposed Design

In this section, we propose a hybrid design for HDFS that supports both socket and RDMA-based communication. First, we discuss the key components of the hybrid design and then describe the associated challenges and RDMA-based communication over InfiniBand. We concentrate on *HDFS Write* because it is more network intensive. Our design extends the existing HDFS and uses UCR for InfiniBand communication via the JNI interfaces.

A. New Components in the Hybrid Architecture

In the hybrid design, we have mainly modified the communication part, keeping the existing HDFS architecture intact. We have applied our modifications to the DFSClient and the

DataNode server. The major components that we have added in the DFSClient side are:

Connection: In the hybrid design, we have introduced a new Java object called Connection. We use UCR as the communication library. As indicated in Section II-E, UCR is an end-point based library. An end-point is analogous to socket connection. The Connection object is responsible for providing the end-point for UCR communication. We maintain a pre-allocated buffer for each Connection object to avoid intermediate data copies between JNI layer and UCR library. The size of this buffer is limited to the HDFS packet-size in order to keep low memory footprint.

The JNI Adaptive Interface: UCR library is implemented in native C code. JNI interface enables the Java code to make use of the UCR library functions for communication.

RDMADataStreamer: Data packets that are put into the DataQueue are sent by the RDMADataStreamer. The RDMADataStreamer runs as a daemon in the RDMA Enabled DFSClient; the RDMA Enabled DFSClient redirects the client to appropriate sender threads (RDMADataStreamer or DataStreamer) depending on whether it wants to send data via RDMA or Java socket. The RDMADataStreamer gets the data packets from the DataQueue and puts the corresponding ByteBuffer into the buffer registered for this particular Connection. The data is then sent over RDMA by invoking the UCR library functions via the JNI interface.

RDMAResponseProcessor: The RDMAResponseProcessor receives acknowledgments for the sent packets over RDMA.

The HDFS DataNode server side is extended by adding different components like:

RDMADataXceiverServer: This is a listener thread which waits for connection requests over RDMA. When the listener accepts an incoming request, an end-point is created between the client and the server. This end-point is equivalent to a socket-connection and can be used for data transfer between the respective nodes.

RDMADataXceiver: After an RDMA connection is established, the RDMADataXceiverServer spawns an RDMADataXceiver thread associated with that connection. This thread behaves in the same manner as the DataXceiver and receives data over RDMA.

Similar to the client side, Connection object and JNI interface components exist in DataNode side as well.

B. Connection and Buffer Management

The DFSClient sends data blocks to different DataNodes for replication, as advised by the NameNode. Therefore, connections are pre-established with all the DataNodes. A Hash-Table is maintained to store information about these connections. When the client has some data to send to a particular DataNode, it performs a lookup on the Hash-Table to get the required connection. In the DataNode side also, we maintain a pool of connections to be used for replication. These connections are initialized when the DataNode servers start. The end-points are established during replication. Each connection in the pool can be reused once it is free. In the existing HDFS implementation, the DFSClient establishes a new socket connection with the DataNode server each time it wants to send a block to it. The DataNode side creates a new receiver thread for each block. In our design, we have eliminated the overhead of connection establishment for each block by using pre-established connections. Furthermore, a single receiver thread in the DataNode is responsible for handling all the blocks coming from a particular client to that connection. In HDFS, data packets as well as the header are Java objects. However, RDMA uses memory semantics. Therefore, we have used Java direct buffers which allows us to take advantage of zero-copy data transfer. In this design, we have used two-sided send-recv communication for data transfer. Besides, the size of the JNI buffer registered per connection is limited to the HDFS packet-size as this is the maximum amount of data to be transferred at a time using this buffer. This helps in maintaining a low memory-footprint. The connections are also established during system initialization. Therefore, the start-up time should also be negligible for large data-intensive applications.

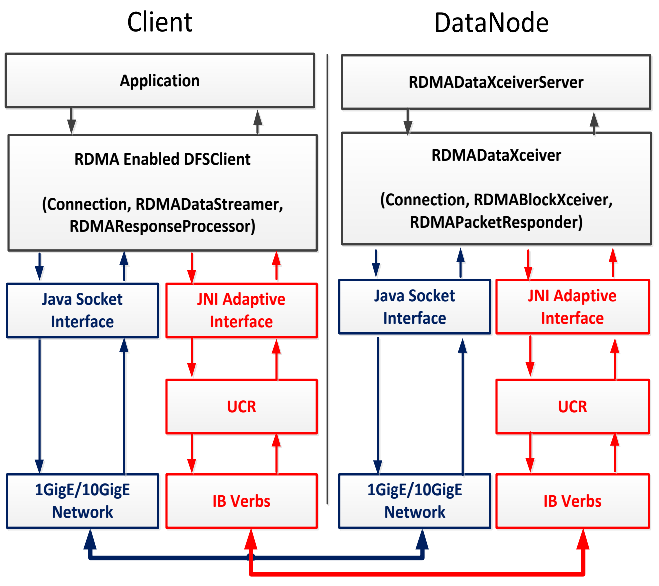


Fig. 3. HDFS-RDMA design with hybrid communication support

C. Communication Flow using RDMA over InfiniBand

The hybrid design supports both RDMA and socket-based communication. Figure 3 illustrates the communication flow in our new design. The RDMA Enabled DFSClient can choose which network to use for an operation. Communication over the socket for an *HDFS Write* follow the same flow as shown in Figure 2. UCR [8] is an end-point based communication model. Two nodes must establish an end-point before they can communicate with each other. When the DFSClient wants to write a file, it first contacts the NameNode over IPC to get a block ID and the list of DataNodes for that block. The block is sent to the first DataNode by the client and then replicated to subsequent DataNodes. The communication between the client and the first DataNode and also among the DataNodes take place over RDMA in the hybrid design. Therefore, the client first checks, if there is already an established end-point with the first DataNode server in the pipeline. If so, the client uses it for data transfer. Otherwise, a new end-point must be created. In the DataNode server side, the RDMADataXceivers monitor their respective endpoints for arrival of data. The acknowledgments are also sent using the endpoint. The RDMAResponseProcessors, on the RDMA Enabled DFSClient, receive the acknowledgments in their corresponding endpoints.

V. Performance Evaluation

In this section, we present the detailed performance evaluations of our RDMA-based design over the existing socket-based implementation of HDFS. We compare the performance of our design with that of 1 GigE, IPoIB and 10 GigE. In HDFS, network performance is sensitive to different HDFS parameters (e.g. packet-size). As different interconnects/protocols support different network bandwidths, the optimal packet-size may also vary with the interconnects/protocols. Therefore, in our experiments, we first determine the optimal packet-size for different interconnects/protocols. After identifying the optimal packet-sizes, we have performed both micro-benchmark level evaluations as well as experiments with TestDFSIO. In this study, we perform the following sets of experiments:

(1) Determining the Optimal Packet-Size for Different Interconnects//Protocols: In this set of experiments, we have varied the HDFS packet-size and found out the optimal value for each interconnect/protocol.

(2) Micro-benchmark Level Evaluations on Different Interconnects: In this set of experiments, we have measured the latency of *HDFS Write* for different file sizes (from 1 GB to 5 GB) on different interconnects, using the optimal packet-sizes obtained in our previous set of experiments.

(3) Experiments with TestDFSIO: In this set of experiments, we have measured the throughput of sequential write operations to HDFS.

In the experiment sets (2) and (3), we have presented results for *HDFS Write*. In most of the practical MapReduce applications, *HDFS Read*s are node local [28]. Thus, we have not focused on *HDFS Read*s. In all our experiments, we have used Hadoop version 0.20.2. The HDFS replication factor was set to three.

A. Experimental Setup

We used two different clusters for our evaluations.

(1) Intel Clovertown Cluster (Cluster A): This cluster consists of 64 compute nodes with Intel Xeon Dual quad-core processor nodes operating at 2.33 GHz with 6GB RAM and PCIe 1.1 interface. Each node is equipped with a ConnectX DDR IB HCA (16 Gbps data rate). 16 of the compute nodes have Chelsio T320 10GbE Dual Port Adapters with TCP Offload capabilities. Each node runs Red Hat Enterprise Linux Server release 5.5 (Tikanga), with kernel version 2.6.30.10 and OpenFabrics version 1.5.3. The IB cards on the nodes are interconnected using a 144 port Silverstorm IB DDR switch, while the 10 GigE cards are connected using a Fulcrum Focalpoint 10 GigE switch. We have performed the experiments with four DataNodes in this cluster.

(2) Intel Westmere Cluster (Cluster B): This cluster consists of 144 compute nodes with Intel Westmere series of processors using Xeon Dual quad-core processor nodes operating at 2.67 GHz with 12GB RAM. Each node is equipped with MT26428 QDR ConnectX HCAs (32 Gbps data rate) with PCI-Ex Gen2 interfaces. The nodes are interconnected using a Mellanox QDR switch. Each node runs Red Hat Enterprise Linux Server release 6.1 (Santiago) at kernel version 2.6.32-131 with OpenFabrics version 1.5.3. Cluster B also has 16 dedicated storage nodes with the same configuration, but with 24GB of RAM each. Additionally, each of the storage nodes is equipped with a 300GB OCZ VeloDrive PCIe SSD. This cluster is less than two years old and represents the leading-edge technologies used in commodity clusters. In all our experiments, the HDFS NameNode runs exclusively on a compute node. In this cluster, we have used two different configurations: i) 32-DataNode HDD cluster (32 compute nodes are used as HDFS DataNodes) and ii) 4-DataNode SSD cluster (four storage nodes are used as DataNodes).

In the figures presented in this section, we have mentioned UCR-IB to indicate the RDMA-based design of HDFS. 32 Gbps and 16 Gbps indicates QDR and DDR platforms, respectively.

B. Determining the Optimal Packet-Size for Different Interconnects/Protocols

In this set of experiments, we have varied the HDFS packet-size from 64 KB to 1 MB and measured the file write times for different file sizes (from 1 GB to 5 GB) using a simple *HDFS Write* benchmark. We have performed these experiments for 1 GigE, IPoIB and RDMA on Cluster B and 1 GigE, IPoIB, 10 GigE and RDMA on Cluster A. For a particular interconnect/protocol, the packet-size, for which we obtain the smallest write latency for most of the file sizes is chosen as the optimal packet-size for the corresponding interconnect/protocol.

Figure 4 shows the optimal packet-sizes for different interconnects/protocols in Cluster B configuration (ii). As it can be observed from the figure, 64 KB packet-size provides minimum latency for most of the file sizes in 1 GigE network. Therefore, 64 KB is chosen as the optimal packet size for 1 GigE network. Similarly, the optimal packet-size for IPoIB and RDMA are 128 KB and 512 KB, respectively. We have done similar experiments for Cluster B configuration (i) and Cluster A also. Because of space limitations, we have not included these graphs. The optimal packet-size values determined for different interconnects/protocols are depicted in Table I. In all these experiments, HDFS block-size is kept fixed at 64 MB. The experiments are run using both HDD and SSD as the underlying storage medium in the DataNode servers. The optimal packet-sizes identified for both HDD and SSD are the same. But, for HDD, the optimal packet-

File Size (GB)	Cluster A				Cluster B		
	1GigE	10GigE	IPoIB	UCR-IB	1GigE	IPoIB	UCR-IB
1	64KB	64KB	128KB	512KB	64KB	128KB	512KB
2	64KB	64KB	128KB	512KB	64KB	128KB	512KB
3	64KB	64KB	128KB	512KB	64KB	128KB	512KB
4	64KB	128KB	128KB	512KB	64KB	128KB	512KB
5	64KB	64KB	128KB	256KB	64KB	128KB	512KB

TABLE I
Optimal Network Packet-Sizes for Different Interconnects/Protocols in both Clusters

	1GigE	IPoIB (32Gbps)	UCR-IB (32Gbps)
Communication	325 ms	77 ms	47 ms
Processing	143 ms	139 ms	112 ms
I/O	299 ms	131 ms	110 ms

TABLE II
Split-up Times for a Single Block Transmission in HDFS

sizes primarily benefit the communication; the bottleneck is still in the I/O. Therefore, even though 64 KB, 128 KB and 512 KB are the optimal packet-sizes for 1 GigE, IPoIB and RDMA respectively, we do not observe much variation in the file write times for different packet-sizes in a particular interconnect/protocol.

C. Micro-benchmark Level Evaluations on Different Interconnects

In our study, we have performed a comprehensive profiling to understand HDFS behavior and determined that, while writing a 64 MB block to HDFS, time is spent on three main parts: (1) Communication, (2) Processing, and (3) I/O. We have measured the time taken by each of these parts for a 64 MB block write by placing timers inside HDFS code. Table II shows these three times measured in different interconnects of Cluster B using HDD. These numbers are measured using a single DataNode, with a replication factor of one.

The use of RDMA over InfiniBand improves the communication time. Here the I/O time is the total time required to flush all the data packets for the block to the file system. Since the packet-sizes used for IPoIB and RDMA are 128 KB and 512 KB, respectively, we are performing aggregated I/O operations in these two cases compared to 1 GigE. As a result, each of these interconnects can gain in terms of I/O time to some extent. After receiving each packet, HDFS performs some processing on it. As part of processing, some Java functions are called per packet basis. Since the number of packets is reduced in our design, we gain in terms of processing time also.

We have designed a micro-benchmark that measures and reports HDFS file write times. Using this micro-benchmark, we have measured the latency of writing files ranging from 1 GB to 5 GB. During the file write operation, we have measured the communication time also by placing timers inside the Java code of HDFS. We have also verified the communication times by designing a micro-benchmark that mimics the communication pattern of HDFS. We have run these experiments in both Cluster A and Cluster B. In each of these experiments, we have used the optimal packet-sizes obtained in our previous experiments for different interconnects.

Figure 5(a) shows the file write times for different file sizes over 1 GigE, IPoIB, 10 GigE and RDMA in Cluster A. We can observe that the RDMA-based design outperforms 10 GigE by 14% and IPoIB by 20% for 5 GB file size. Figure 5(b) shows the file write times for different file sizes over 1 GigE, IPoIB, and RDMA in Cluster B. As we can observe from this figure, the RDMA-based design outperforms the socket-based design for all the file sizes and the overall gain in terms of latency is 15% over IPoIB for 5 GB file size. In this experiment, HDD is used as the underlying storage device in the DataNodes.

Figure 6(a) shows the file write times in Cluster A using SSD. Due to decrease in I/O times with SSD, the total file write times are less compared to those in Figure 5(a). Our design achieves an improvement of 16% over 10 GigE and 25% over IPoIB for 5 GB file size. Figure 6(b) shows the file write times for different file sizes in three different interconnects of Cluster B configuration (ii). From this figure, we observe that, our design improves the write latency by 12% over IPoIB for 5 GB file size.

Figure 7 shows the times spent in communication during *HDFS Write* of different file sizes, measured over different interconnects in Cluster B with 32 DataNodes with HDD. The new design gains an improvement of 30% over IPoIB and 87% over 1 GigE in terms of communication time. We have measured the communication times in Cluster B with four SSD nodes also and observed similar gains in communication times. This clearly demonstrates the capability of the native RDMA-based design to provide low-latency data transmission.

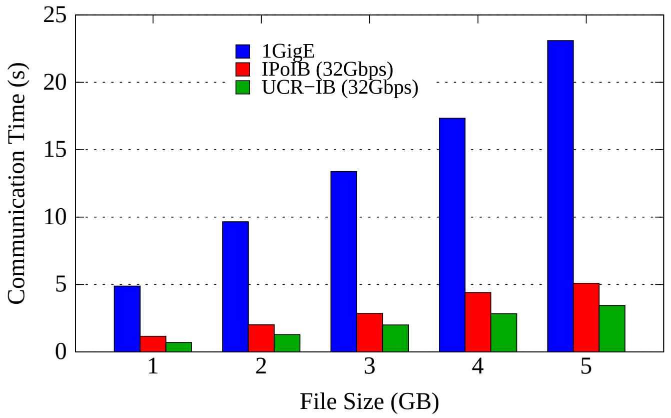
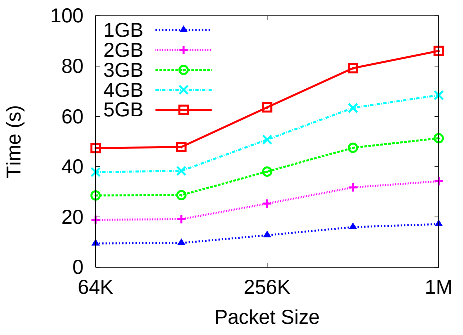


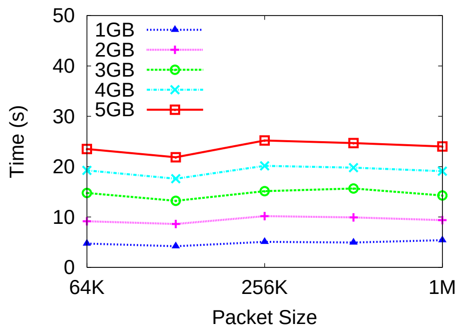
Fig. 7. Communication times in HDFS in Cluster B with 32 DataNodes

D. Experiments with TestDFSIO

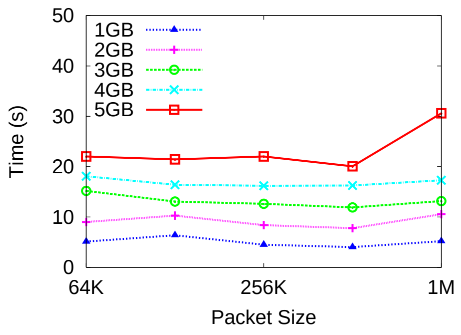
DFSIO is a file system MapReduce benchmark in Hadoop that measures the I/O performance of HDFS. In this benchmark, each map task opens one file for sequential read/write and the Hadoop job measures the data I/O size and the execution time for writing a particular sized file [7]. There is only a single reduce task which aggregates the results from different map tasks running in this benchmark. In our experiments, we run DFSIO write with file sizes ranging from



(a) 1GigE

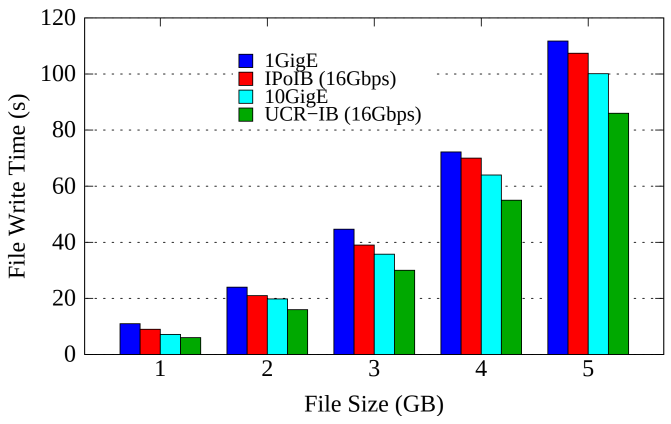


(b) IPoIB (32Gbps)

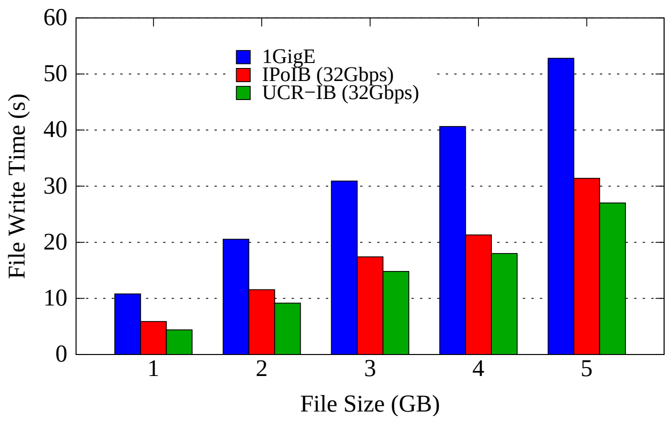


(c) UCR-IB (32Gbps)

Fig. 4. HDFS optimal network packet-size evaluation for different file sizes - Cluster B

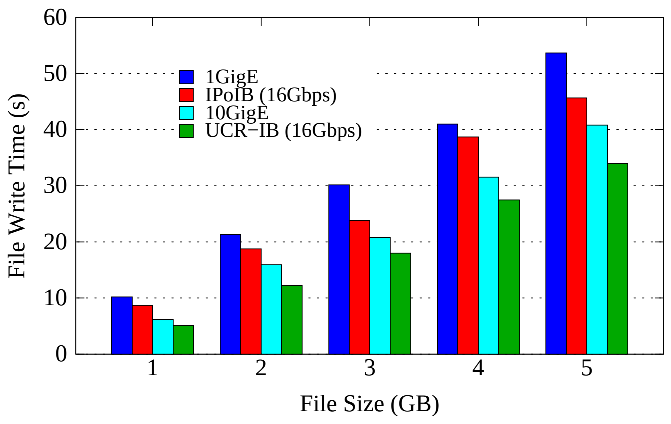


(a) Total file write times in HDD (Cluster A with 4 DataNodes)

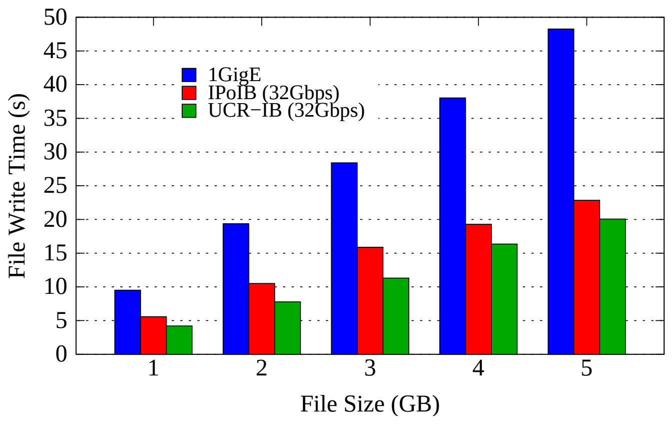


(b) Total file write times in HDD (Cluster B with 32 DataNodes)

Fig. 5. Microbenchmark evaluation in different clusters for HDD



(a) Total file write times in SSD (Cluster A with 4 DataNodes)



(b) Total file write times in SSD (Cluster B with 4 DataNodes)

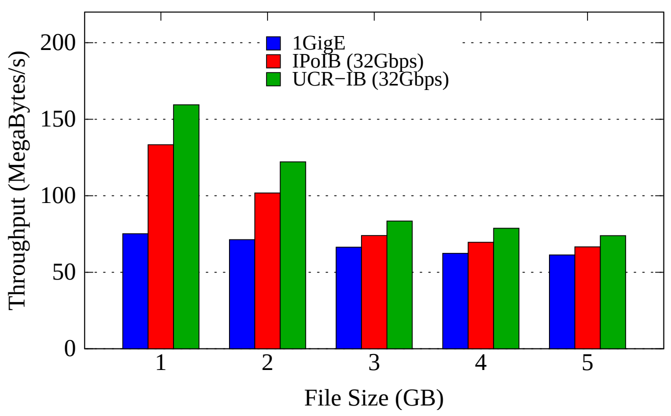
Fig. 6. Microbenchmark evaluation in different clusters for SSD

1 GB to 5 GB. We have performed these experiments in Cluster B. Figure 8(a) shows the throughput results of DFSIO in HDD. We have done this experiment with 32 DataNodes.Here we can see that our design outperforms IPoIB by 12% for 5 GB file size. Figure 8(b) shows the throughput results of DFSIO in SSD for three different interconnects of Cluster B with four DataNodes. From this figure we can observe that, our performance gain over IPoIB is 10% for 5 GB file size. Therefore, the UCR-based design is able to achieve better throughput compared to the socket-based design.

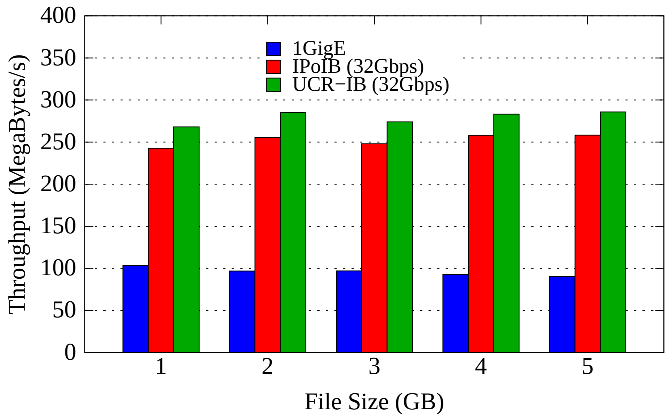
VI. Benefit of RDMA-based HDFS in HBase

In this section, we evaluate HBase *Put* operation with the underlying file system as HDFS. The most useful use-case for HBase with HDFS is the bulk writes or updates.

With large amount of data insertion, HBase eventually flushes its in-memory data to the HDFS. HBase has a MemStore which holds the in-memory modifications to any particular region of data. MemStore triggers a flush operation when it reaches its size limit (default is 64 MB). During the flush, HBase writes the MemStore to HDFS as an HFile instance. These HFiles are written per flush of MemStore and a compaction of all these store files is needed when the number



(a) TestDFSIO in HDD (Cluster B with 32 DataNodes)



(b) TestDFSIO in SSD (Cluster B with 4 DataNodes)

Fig. 8. TestDFSIO benchmark evaluation in different interconnects

of store files reaches a particular threshold (default is three) for any one HStore. Although MemStore flush to HStore, HStore files' compaction and HBase update to MemStore all happen concurrently in different threads, high operational latency for *HDFS Writes* can affect the overall performance of HBase *Put* operation. HBase also has an HLog instance which keeps flushing its log data to HDFS. These HLogs are the basis of HBase data replication and failure recovery, and thus must be kept in HDFS.

With the new design of HDFS, we conducted experiments to measure the overall *Put* latency in HBase. These experiments are run on Cluster B on a QDR platform (32Gbps) and we use HDD as storage for HDFS. A modified version of workload A (100% update) of YCSB (mentioned in Section II) has been used as our benchmark. Figure 9 shows these results for HBase *Put* operation of 1 KB message size with a single region server. For insertion of 360 K records, an average latency of 204 μs is achieved whereas in IPoIB the average latency is 252 μs. The throughput achieved by the RDMA-based design is 4.41 $Kops/sec$, whereas IPoIB obtains a throughput of 3.63 $Kops/sec$. Our HDFS design gets an overall performance improvement of around 20% in both average latency and throughput.

In order to have an increased number of DFSClients in our experiments, we have increased the number of region servers to 32 using configuration (i) of Cluster B. With 32 region servers, we experience similar performance improvement for the new HDFS design on QDR platform (32 Gbps). For 480 K records insertion, each of 1 KB message size, an average latency of 201 μs is achieved, which is 26% less than that of IPoIB (272 μs). The throughput for 480 K record insertion is 4.42 $Kops/sec$, which is also 24% higher than that of IPoIB (3.35 $Kops/sec$). Figure 10 illustrates these results.

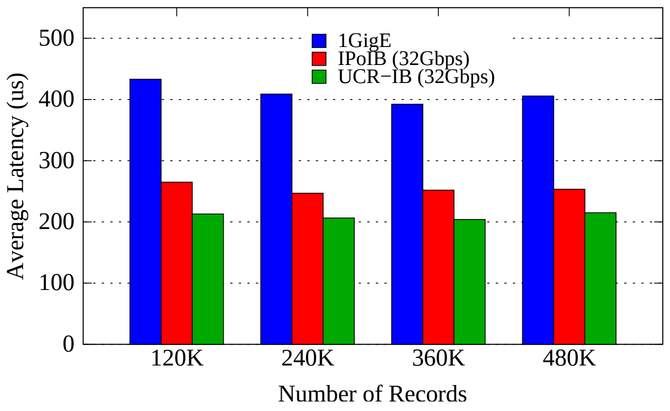
VII. Related Work

The Hadoop Filesystem has taken a prominent role in the Hadoop ecosystem. Prior work in literature has already analyzed the HDFS architecture for drawbacks and limitations. For instance, Shvacho [29] studied correlation between HDFS design goals and the feasibility of achieving them with current system architecture. He takes a retrospective view of the set-targets and the factors that limit growth. He also provides several conclusions that highlights dependencies between the components in HDFS that limit growth - Namespace limitations, cluster sizes, internal loads, etc. Along similar lines, Shafer et al. [30] analyze the performance of HDFS and identifies three critical issues - architectural bottlenecks that cause delays in scheduling new tasks, portability limitations posed by the Java implementation and assumptions about storage management. They investigate the causes of these issues and suggest potential improvements to the HDFS architecture that will help achieve a better balance between performance and portability.

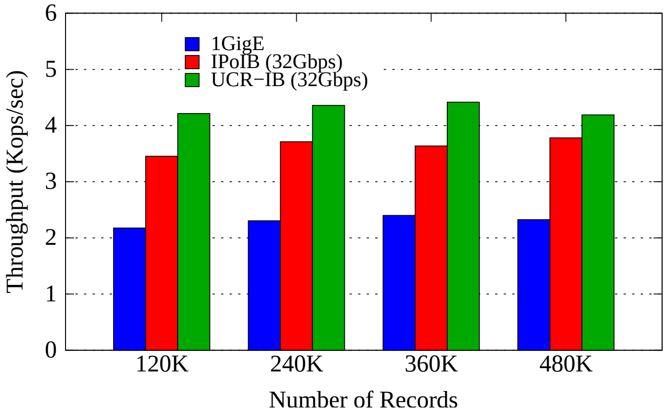
A number of other solutions have been proposed to address the issues identified by Shafer et al. and Shvacho. A significant majority of these address the Namespace limitation [31–34] posed by the native HDFS architecture that employs a single-node namespace. Having a single NameNode constrains the capacity of HDFS namespace to the amount of physical memory available to that node. The proposed solutions consider variations of a distributed meta-data management scheme that can alleviate the load on a stand-alone NameNode.

Another leg of research that looks at improving the performance of HDFS explores the interoperability between HDFS and certain existing POSIX-compliant parallel filesystems such as PVFS [35], Ceph [36], GPFS [37], GFarm [38], etc. These studies either provide interfaces that the Hadoop environment can use to replace HDFS with the corresponding filesystem, or propose HDFS-specific optimizations in their filesystem which makes it compatible for use in the Hadoop ecosystem. A major challenge that was addressed by these studies was to map HDFS semantics to POSIX semantics, in the context of their respective architectures.

In HPC field, RDMA has been used to speed up file systems I/O performance. Wu et. al designed the PVFS over InfiniBand to improve PVFS I/O performance [39]. Ouyang et. al designed a RDMA based job migration framework to use RDMA improve large-sized jobs recovery [40]. Yu et. al proposed a RDMA-capable storage protocols on wide area network to speedup NFS performance [41]. These studies

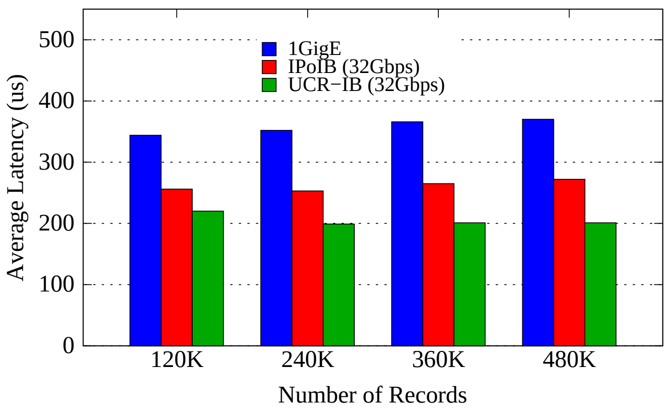


(a) *Put* latency

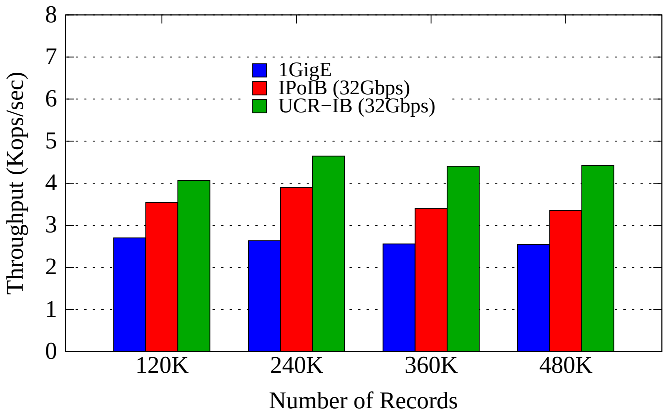


(b) *Put* throughput

Fig. 9. YCSB evaluation with a single region server



(a) *Put* latency



(b) *Put* throughput

Fig. 10. YCSB evaluation with multiple (32) region servers

illustrate that RDMA can benefit the traditional distributed and parallel file systems. We share similar objectives with these research directions and investigate the benefit of RDMA in the Hadoop environment.

An RDMA-based high-performance design of HBase over InfiniBand was presented in [6] where the HBase data-query performance was improved by optimizing the communication cost using RDMA. The memory-block semantics supported by RDMA-capable networks were mapped to the object transmission primitives used in HBase.

The study by Wang et al. [42] reveals that the `Merge` operation in MapReduce can be accelerated by exploiting the benefits of RDMA in such a manner that the data does not have to be copied to disk. They have also developed a shuffle-merge-reduce pipeline that works in conjunction with the RDMA-based merge.

In our prior work [7], we examined the impact of high-speed interconnects such as 1GigE, 10GigE and InfiniBand (IB) using protocols such as TCP, IP-over-IB (IPoIB) and Sockets Direct Protocol (SDP), on HDFS performance. Our findings also revealed that these faster interconnects make a larger impact on the performance if SSDs are used in DataNodes, as this reduces the local I/O costs. The findings in this work has lead us to further investigate the fundamental communication overheads in native HDFS and redesign it to leverage advanced features offered by InfiniBand, such as RDMA.

VIII. Conclusion and Future Work

In this study, we propose a hybrid design for HDFS that incorporates communication over conventional socket as well as RDMA over InfiniBand. The new design is able to provide low-latency and high throughput for *HDFS Write* operations as it leverages the RDMA capability of high performance network like InfiniBand. Our design achieves 30% gain in communication time and 15% gain in overall write time over IPoIB on QDR platform during *HDFS Write* for different file sizes. The new design also increases the throughput of DFSIO by 12% over IPoIB. For HBase, the *Put* operation performance is improved by 26% with our new design.

We plan to continue our studies along this direction. We plan to identify the higher level architectural bottlenecks in HDFS and propose high performance designs. We also plan to come up with a highly scalable design of HDFS with improved communication performance and investigate on faster recovery in case of DataNode failures. Further, we would like to evaluate the performance benefits of other cloud computing middlewares which use HDFS as the filesystem with these new designs.

References

[1] 2011 IDC Digital Universe Study, http://www.emc.com/leadership/programs/digital-universe.htm.

[2] J. Appavoo, A. Waterland, D. Da Silva, V. Uhlig, B. Rosenburg, E. Van Hensbergen, J. Stoess, R. Wisniewski, and U. Steinberg, "Providing A Cloud Network Infrastructure on A Supercomputer," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 385–394.

[3] Greenplum Analytics Workbench, http://www.greenplum.com/news/greenplum-analytics-workbench.

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

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

[6] J. Huang, X. Ouyang, J. Jose, M. W. Rahman, H. Wang, M. Luo, H. Subramoni, C. Murthy, and D. K. Panda, "High-Performance Design of HBase with RDMA over InfiniBand," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS'12)*.

[7] S. Sur, H. Wang, J. Huang, X. Ouyang, and D. K. Panda, "Can High Performance Interconnects Benefit Hadoop Distributed File System?" in *Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds, in Conjunction with MICRO 2010*, Atlanta, GA, 2010.

[8] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. W. Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, and D. K. Panda, "Memcached Design on High Performance RDMA Capable Interconnects," in *International Conference on Parallel Processing (ICPP)*, Sept 2011.

[9] The Apache Software Foundation, "The Apache Hadoop Project," `http://hadoop.apache.org/`.

[10] J. Jose, M. Luo, S. Sur, and D. K. Panda, "Unifying UPC and MPI Runtimes: Experience with MVAPICH," in *Fourth Conference on Partitioned Global Address Space Programming Model (PGAS)*, Oct 2010.

[11] Top500 Supercomputing System, http://www.top500.org.

[12] 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.

[13] Open Fabrics Enterprise Distribution, http://www.openfabrics.org/.

[14] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '09, New York, NY, USA, 2009.

[15] B. Fitzpatrick, "Distributed Caching with Memcached," *Linux Journal*, vol. 2004, pp. 5–, August 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1012889.1012894

[16] Apache HBase, "The Apache Hadoop Project," `http://hbase.apache.org/`.

[17] Hadoop Map Reduce, "The Apache Hadoop Project," `http://hadoop.apache.org/mapreduce/`.

[18] Apache Hadoop, `http://hadoop.apache.org/`.

[19] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A Distributed Storage System for Structured Data," in *The Proceedings of the Seventh Symposium on Operating System Desgin and Implementation (OSDI'06)*, WA, November 2006.

[20] B. H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," CACM 13,7 (1970), 422-426.

[21] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI04: Proceedings of the 6th conference on Symposium on Opearting Systems Design and Implementation*. USENIX Association, 2004.

[22] Roy T. Fielding and Richard N. Taylor, "Principled Design of the Modern Web Architecture," ACM Transactions on Internet Technology, Vol. 2, No. 2, May 2002, Pages 115.150.

[23] The Apache Software Foundation, "Apache Avro," `http://avro.apache.org/`.

[24] Apache Thrift, `http://thrift.apache.org/`.

[25] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *The Proceedings of the ACM Symposium on Cloud Computing (SoCC 2010)*, Indianapolis, Indiana, June 10-11 2010.

[26] The Apache Software Foundation, "Apache Cassandra," `http://cassandra.apache.org/`.

[27] B. F. Cooper, R. Ramakrishnan, R. Sears, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS: Yahoo!s Hosted Data Serving Platform," in *34th International Conference on Very Large Data Bases*, 2008.

[28] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Operating Systems Design and Implementation (OSDI)*, 2004.

[29] K. V. Shvachko, "HDFS Scalability: The Limits to Growth," Apr. 2010. [Online]. Available: http://www.usenix.org/publications/login/2010-04/openpdfs/shvachko.pdf

[30] J. Shafer, S. Rixner, and A. L. Cox, "The Hadoop Distributed Filesystem: Balancing Portability and Performance," in *The Proceedings of the Internation Symposium on Performance Analysis of Systems and Software (ISPASS'10)*, White Plains, NY, March 28-30 2010.

[31] G. Mackey, S. Sehrish, and J. Wang, "Improving Metadata Management for Small Files in HDFS," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 31 2009-sept. 4 2009.

[32] Y. Gao and S. Zheng, "A Metadata Access Strategy of

Learning Resources Based on HDFS," in *Image Analysis and Signal Processing (IASP), 2011 International Conference on*, oct. 2011, pp. 620 –622.

[33] B. Li, Y. He, and K. Xu, "Distributed Metadata Management Scheme in Cloud Computing," in *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, oct. 2011, pp. 32 –38.

[34] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, "Hadoop High Availability through Metadata Replication," in *Proceedings of the first international workshop on Cloud data management*, ser. CloudDB '09. New York, NY, USA: ACM, 2009, pp. 37–44. [Online]. Available: http://doi.acm.org/10.1145/1651263.1651271

[35] W. Tantisiriroj, S. Patil, G. Gibson, S. W. Son, S. Lang, and R. Ross, "On the Duality of Data-Intensive File System Design: Reconciling HDFS and PVFS," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, nov. 2011.

[36] C. Maltzahn, E. Molina-Estolano, A. Khurana, A. J. Nelson, S. A. Brandt, and S. Weil, "Ceph as a Scalable Alternative to the Hadoop Distributed File System," Aug. 2010. [Online]. Available: {http://static.usenix.org/publications/login/2010-08/openpdfs/maltzahn.pdf}

[37] K. Gupta, R. Jain, H. Pucha, P. Sarkar, and D. Subhraveti, "Scaling Highly-Parallel Data-Intensive Supercomputing Applications on a Parallel Clustered Filesystem," in *The SC10 Storage Challenge*.

[38] S. Mikami, K. Ohta, and O. Tatebe, "Using the Gfarm File System as a POSIX Compatible Storage Platform for Hadoop MapReduce Applications," in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, ser. GRID '11, 2011.

[39] J. Wu, P. Wyckoff, and D. K. Panda, "PVFS over InfiniBand: Design and Performance Evaluation," in *The Proceedings of the 32nd International Conference on Parallel Processing (ICPP 2003)*, Kaohsiung, Taiwan, October 6-9 2003.

[40] X. Ouyang, S. Marcarelli, R. Rajachandrasekar, and D. K. Panda, "RDMA-based Job Migration Framework for MPI over InfiniBand," in *The Proceedings of the International Conference on Cluster Computing (Cluster 2010)*, Heraklion, Greece, September 20-24 2010.

[41] W. Yu, N. S. V. Rao, and J. S. Vetter, "Experimental Analysis of InfiniBand Transport Services on WAN," in *The Proceedings of the IEEE International Conference on Networking, Architecture, and Storage (NAS 2008)*, Chongqing, China, June 12-14 2008.

[42] Y. Wang, X. Que, W. Yu, D. Goldenberg, and D. Sehgal, "Hadoop Acceleration through Network Levitated Merge," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011.