Designing Virtualization-aware and Automatic Topology Detection Schemes for Accelerating Hadoop on SR-IOV-enabled Clouds

Shashank Gugnani, Xiaoyi Lu, Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH USA 43210
Email: {gugnani.2, lu.932, panda.2}@osu.edu

***Abstract*—Hadoop is gaining more and more popularity in virtualized environments because of the flexibility and elasticity offered by cloud-based systems. Hadoop supports topology-awareness through topology-aware designs in all of its major components. However, there exists no service that can automatically detect the underlying network topology in a scalable and efficient manner, and provide this information to the Hadoop framework. Moreover, the topology-aware designs in Hadoop are not optimized for virtualized platforms. In this paper, we propose a new library called Hadoop-Virt, based on RDMA-Hadoop, which provides with an automatic topology detection module and virtualization-aware designs in Hadoop to fully take the advantage of virtualized environments. Our experimental evaluations show that Hadoop-Virt delivers upto 34% better performance in the default execution mode and upto 52.6% better performance in the distributed mode as compared to default RDMA-Hadoop for SR-IOV-enabled virtualized clusters.**

***Keywords*-Virtualization, Topology-awareness, Hadoop, Big Data**

I. Introduction

The growing demand for Big Data has fueled a revolution in cloud computing [8], [9], [10]. Enterprise users, being the main producers and consumers of Big Data, experience highly variable workloads. For these users, cloud computing offers attractive solutions in the form of scalability, flexibility and reliability [11]. Thus, the combination of Big Data and cloud computing is extremely relevant these days, especially in the enterprise community.

Until a few years back, most people had concerns about the performance of virtualized platforms. However, over time, and with advancements in virtualization technology, most of these concerns have been resolved. In addition, the introduction of containers [12] has only increased the popularity of cloud computing. One of the biggest challenges in the adoption of cloud computing was virtualized I/O. However, with the introduction of Single-Root I/O Virtualization (SR-IOV) [13], this challenge has been alleviated as well. With SR-IOV, a single PCI Express (PCIe) device can be presented as multiple virtual devices, and each of these virtual device can be allocated to a VM. Many studies [14], [15], [16] have shown that SR-IOV can deliver near-native I/O performance, which is much better than the software-based approaches used earlier [17].

Since many scientific applications are becoming extremely computationally demanding, Big Data systems are becoming increasingly complex and large. As systems become large and distributed, the topology of the cluster becomes increasingly important. Each traversal of a switch (hops) between two nodes adds a certain delay to the communication time. Several studies [18], [19] have shown that topology-aware communication can help reduce network congestion and significantly improve performance. Thus, both topology detection and topology-aware communication are necessary for obtaining the best performance. Hadoop, one of the most popular Big Data stacks, already has support for topology-aware communication through topology-aware designs in all of its major components. However, the designs are not optimized for virtualized platforms. Moreover, there exists no support for automatically detecting the topology and exposing it to the Hadoop framework in SR-IOV-enabled InfiniBand clusters. So, the user has to manually discover the network topology and provide it to Hadoop.

When Hadoop is run in virtualized environments, there is an added overhead of the virtualization layer. Even advancements in virtualization technology cannot fully eliminate this overhead. To obtain best performance, Hadoop itself should be optimized for virtual clusters. There have been several works which try to enhance the performance of Hadoop in virtualized environments. Table I shows the comparison of our paper with related papers. Our work is unique in the sense that it provides with virtualization-awareness in all major components. All other papers provide virtualization-aware designs in only one or a maximum of two components. In addition, most of the other papers don't consider the problem of topology detection. This leads us to the following broad challenges:

1) How can the cluster topology be automatically detected in a scalable and efficient manner and be exposed to the Hadoop framework?
2) How can we design virtualization-aware policies in Hadoop for efficiently taking advantage of the detected topology?
3) Can the proposed policies improve the performance and fault tolerance of Hadoop on virtualized platforms?

*This research is supported in part by National Science Foundation grants #CNS-1419123, #IIS-1447804, and #ACI-1450440.

9781-5090-1445-3/16$31.00

	Virtualization-awareness			Topology detection		RDMA-enabled designs
	YARN	MapReduce	HDFS	Automatic detection	Heterogeneous support	
[1], [2]	×	√	√	×	×	×
[3], [4]	×	√	×	×	×	×
[5]	×	√	√	√	×	×
[6]	×	×	√	×	×	×
[7]	√	×	×	×	×	×
This paper	√	√	√	√	√	√

Table I
COMPARISON WITH RELATED WORKS

Apache Hadoop [6] is an open-source Big Data stack, which has become extremely popular in recent years. It has become the primary package used for Big Data analytics. Remote Direct Memory Access (RDMA) is a feature available in modern networking interconnects like InfiniBand [20]. RDMA allows one node to directly access the memory of a remote node, without any CPU involvement from the remote node. RDMA-Hadoop [21] is a publicly available stack based on Apache Hadoop which provides with RDMA-enhanced designs in Hadoop for RDMA-enabled clusters.

In this paper, we propose a new library called Hadoop-Virt, based on RDMA-Hadoop, which has virtualization-aware components and support for automatic topology detection through a specialized topology detection module, for taking full advantage of virtualized systems. To summarize, the main contributions of this paper are as follows:

1) Virtualization-aware policies for map task scheduling and container allocation
2) Topology detection module for automatic topology detection of Hadoop clusters, including detection of VM placement on physical nodes for virtualized clusters
3) Support for heterogeneous clusters (with bare-metal nodes and VMs) for the policies created above and within Hadoop itself

Our experimental evaluations show that Hadoop-Virt delivers upto 34% better performance in the default execution mode and upto 52.6% better performance in the distributed mode as compared to default RDMA-Hadoop for SR-IOV-enabled virtualized clusters.

The rest of this paper is organized as follows. Section II presents an overview of InfiniBand and RDMA-Hadoop. Section III describes the proposed designs. Section IV presents our performance evaluation results. We discuss related work in Section V and conclude our work in Section VI.

II. BACKGROUND

A. InfiniBand

InfiniBand [20] is a networking interconnect that delivers high performance and is widely used in supercomputers for high performance computing. The latest TOP500 [22] rankings released in June 2016 show that more than 40% of the top 500 supercomputers use InfiniBand as their networking interconnect. Remote Direct Memory Access (RDMA) is one of the main features of InfiniBand. Through RDMA, a node can directly access the CPU memory of a remote node without any involvement from the remote node. InfiniBand communication is processed in userspace and carried out in a 'zero-copy' manner. InfiniBand uses hardware offload for all protocol processing, resulting in high bandwidth and low latency communication.

B. RDMA-Hadoop

RDMA-Hadoop [21] is a publicly available library built on Apache Hadoop that can be used to exploit the advantages of RDMA on RDMA-enabled clusters for Big Data applications. RDMA-Hadoop provides with high performance designs for HDFS [23], MapReduce [24], and Remote Procedure Call (RPC) [25] components which are optimized for RDMA-enabled clusters. The HDFS plugin can be operated in multiple modes: HHH - the default mode, HHH-M - with support for in-memory I/O operations, and HHH-L - for use with Lustre filesystem. In addition, it has policies which make efficient use of heterogeneous storage devices like SSD, HDD, RAM Disk, and Lustre. The MapReduce plugin has an advanced design that features RDMA-based shuffle, pre-fetching of map output, and optimized overlapping of different MapReduce stages. The RPC plugin features RDMA-based data transmission, and JVM-bypassed buffer management with smart buffer allocation.

III. PROPOSED DESIGN

In this work, we propose a new library called Hadoop-Virt. This package has virtualization-aware components for virtual Hadoop clusters. The different components of the proposed design are described in detail below.

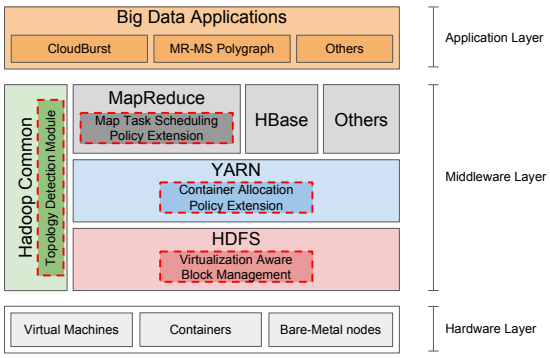


Figure 1. Overview of the Hadoop-Virt Architecture

A. Overview of Hadoop-Virt

Hadoop-Virt has virtualization-aware components in all the four main Hadoop components - HDFS [26], YARN [27], MapReduce [28], and Hadoop Common. Figure 1 shows the Hadoop-Virt architecture. The components highlighted in

dashed boxes are the additions we have made to the Hadoop framework. We provide with a topology detection module in Hadoop Common. We propose extensions to the container allocation policy in YARN and map task scheduling policy in MapReduce to add a virtualization layer. We also provide with virtualization-aware block management in HDFS. Thus, in Hadoop-Virt, all components are optimized for virtualized clusters.

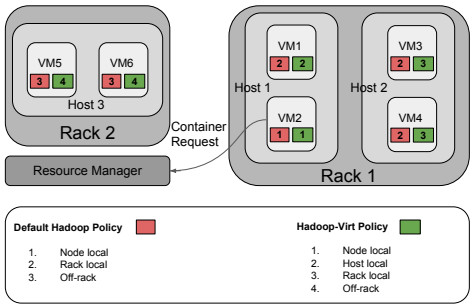


Figure 2. YARN Container Allocation Policy

B. Virtualization Aware Policies

In SR-IOV-enabled virtualized environments, VMs co-located on the same host can communicate with each other through loopback mechanism. Since loopback is faster than having to go through network switches, communication between co-located VMs/containers will be faster than communication between VMs located on different hosts. Thus, to make network communication more efficient, the number of Host local communications should be maximized, where Host local communication means communication between co-located VMs. We define "Host" as the hypervisor or bare-metal node on which a VM is placed and "Host local" (for a VM) to signify VMs which are on the same Host (hypervisor). The Host name will give us an indication of which VMs are on the same hypervisor, since their Host names will be the same.

To take advantage of the benefits of Host local communication, we propose virtualization-aware policies. These are described in detail below.

1) YARN Container Allocation: Container allocation plays an important role in increasing the locality of communication. For example, to schedule a map task on a node, we first need to allocate a container on that node. Thus, map task scheduling depends on container allocation for locality in communication. Locality for container allocation is with regard to the node on which a container is requested. By default, YARN will allocate containers in the order - Node local, Rack local, and Off-rack. So, if we are running Hadoop in a virtualized environment, and we are not able to allocate a container on a node local node, it will be allocated on either a rack local node or an off-rack node. However, this is not optimal, since we didn't consider Host local nodes before Rack local nodes, which will provide with better locality. Thus, we extended this policy to add a Host local layer. The new policy allocates container in the order - Node local, Host local, Rack local, and Off-rack. Figure 2 shows the default and proposed container allocation policies. We can observe that if the ResourceManager fails to allocate a container on VM 2, it will be allocated on either VM 1, 3, or 4 under the default policy. Thus, there is a good chance that the container will be allocated on VM 3 or 4, which are not on the same host as VM 2, and thus not optimal for locality. The proposed policy will allocate the container on VM 1 if it fails to allocate the container on VM 2.

2) Map Task Scheduling: Map task scheduling tries to ensure that the map task is scheduled on a node which is as close (in terms of locality) to the node where the data is placed. This locality aware scheduling is important to reduce network traffic and reduce communication time. Similar to the container allocation policy, the map task scheduling will assign tasks in the order - Node local, Rack local, and Off-rack. This is not optimal, since we didn't consider Host local nodes before Rack local nodes, which will deliver better communication locality. The new policy assigns map tasks in the order - Node local, Host local, Rack local, and Off-rack. This ensures that the communication pattern and cost is optimal for virtualized environments.

C. Automatic Topology Detection

Topology-awareness is important in the execution of applications on large scale clusters. Hadoop has support for topology-aware execution. However, the main problems with the existing topology-aware designs in Hadoop are twofold. Firstly, there is no support for the automatic detection of topology of Hadoop clusters. So, the user has to create a topology file manually and activate the topology-aware designs to fully utilize the benefits offered by these designs. Furthermore, the existing topology-aware designs don't offer any support for virtualized clusters. Secondly, modern virtualized clusters support automatic VM migration for load-balancing and energy saving. Hence, the topology of a virtual Hadoop cluster may change at any time. Having up-to-date topology information is essential to ensure that the internal topology-aware algorithms deliver best performance. Hadoop supports dynamic topology detection through a topology detection script (to be provided by the user) which is executed each time the topology information is needed. However, this is not efficient since the topology detection script does take a considerable amount of time to execute.

Thus, there is a need for virtualization-aware policies and automatic topology detection to enable users to reap the benefits of topology-awareness without the need to do any work manually. We propose a topology detection module for Hadoop over InfiniBand networks, which will not only detect the overall network topology, but will also get information about the VM placement (i.e. which VMs are co-located on the same host). This information will be useful when using the virtualization-aware policies proposed earlier. So, our topology detection module should be able to find information about the rack a node is placed in and the host on which the VM/Container resides. To get this information and create a topology file that can be used as input to Hadoop-Virt, we

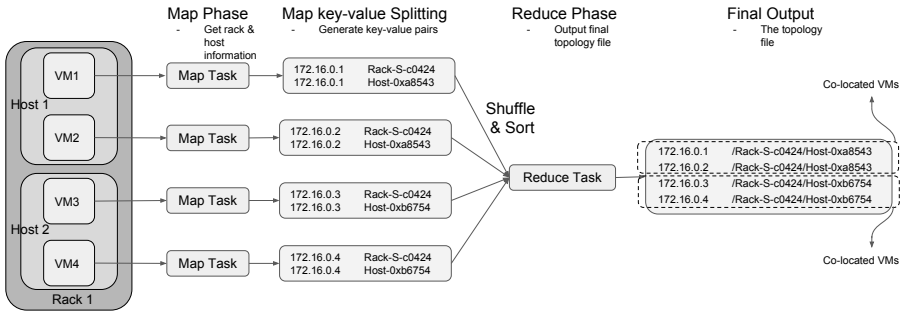


Figure 3. MapReduce-based Utility for Topology Detection

propose two methods - a topology detection script and a MapReduce-based utility for topology detection. Our topology detection module is currently implemented only for InfiniBand networks. However, we implemented the InfiniBand specific part of the module as a plugin so that plugins for other network types can be easily created and inserted into the module. The detailed description of our topology detection script and MapReduce-based utility is presented below.

1) Topology Detection Script: To detect the InfiniBand network topology, we used the *ibnetdiscover* command that is provided with the InfiniBand OFED package [29]. While this command will provide us with the network topology of the InfiniBand network, it will just provide us with the details about the bare-metal hosts and not the VMs. To alleviate this issue, we used the GUID (Global Unique Identifier) of the HCA (Host Channel Adapter) to parse the information about the network connectivity of the VMs. Since the GUID is associated with the HCA, it is common for the VMs and the Host for those VMs. Hence, we can use the GUIDs to find the network topology of the VMs.

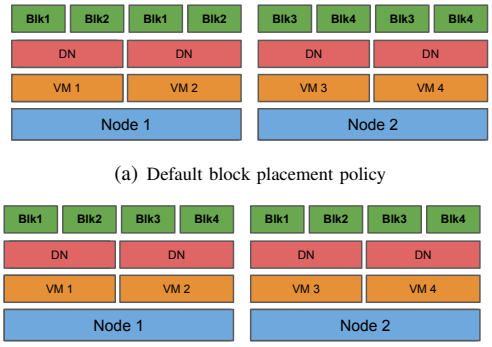
In addition to finding the network topology, we also need to find the VM placement information. We directly use the GUID of the HCA as the Host name of each node, since the GUID is common for all VMs on the same node.

2) MapReduce-based Utility for Topology Detection: Although the topology detection script can successfully get the topology information of the Hadoop cluster, it is not scalable or fault tolerant. This is because it is run on a single node and is based on ssh. Thus, we propose a scalable MapReduce-based utility for detecting the cluster topology. For the MapReduce utility, we run one map task on each slave of the Hadoop cluster and one reduce task in total. For the input of the map tasks, we create dummy files of negligible size. The map task gets the Rack and Host name in the same way as the topology detection script. It then outputs two key-value pairs, one for the Rack and one for the Host, where the key is the IP address of the slave and the value is the Rack/Host name. The reduce task combines the key-value pairs in such a way that the output will directly give us the desired topology file. Figure 3 shows the working of the application. Since the utility is based on the MapReduce framework, it is inherently scalable and fault tolerant. In addition, the final topology file is also stored on HDFS, increasing the level of fault tolerance.

We propose two modes of operation for our topology detection module - static and dynamic. In the static mode, we provide the topology detection script and MapReduce application to the user. The user can run either the script or the utility, and it will create a topology file, which is then used by Hadoop for getting the topology information. In the dynamic mode, there are three sub-modes. The first sub-mode uses the topology detection script, the second uses the MapReduce-based utility, and the third uses both. In the first sub-mode, the topology detection script is run the first time the topology information is needed and this information is then cached. We then used this cached information for further topology requests, until we receive a certain number of topology requests (which is configurable via a parameter), after which we run the topology detection script again. Since the script is not run each time, it should have negligible overhead. In the second sub-mode, the topology detection module is run in a separate thread. This thread wakes up every few minutes and checks the number of jobs running on the Hadoop cluster. If the cluster is idle, then it runs the MapReduce application for detecting the cluster topology. Since we only run the utility when the cluster is idle, it should not affect the performance of any applications running on the cluster. In the third sub-mode (or Hybrid mode), we dynamically select from the first and second sub-modes, based on which mode is the best for the number of nodes in the cluster. From our evaluation results (Section IV-F) we discovered that the script is more efficient for a small sized cluster, whereas the MapReduce application is better for large sized clusters. Thus, we set a threshold value (configurable via a parameter) for the number of nodes in the cluster, below which the script will be run, and above which the MapReduce application will be run. The default value of this parameter is tuned for our experimental testbed. However, this can easily be tuned for any cluster.

D. Virtualization Aware Block Management

For the replication of a block, HDFS places the first block on the primary node, the second block on a node on a different rack than the first node, and the third block on a node on the same rack as the primary node. With this policy, it is possible that HDFS places blocks on VMs which are on the same physical host, as shown in Figure 4(a). This is not ideal from a fault tolerance perspective, since a failure of the physical host will result in loss of both the replicas. Hadoop already has



(a) Default block placement policy

(b) Virtualization-aware block placement policy

Figure 4. Block placement scenarios for different block placement policies

support for virtualization-aware block management (VABM), which can be enabled using a parameter. VABM ensures that replicas of the same block are not placed on VMs on the same physical host. Figure 4 shows a block placement scenario for 2 blocks on a 4 VM, 2 physical node Hadoop cluster with a replication factor of 2. With the default placement policy scenario, if one of the physical nodes crashes, we will loose two blocks, since both replicas of each block are on the same physical host. But, with the virtualization-aware policy, replicas of the block are not placed on co-located VMs. Hence, failure of any physical node will not lead to any data loss. VABM ensures that the fault tolerance level of virtual cluster is the same as bare-metal clusters. However, VABM is not compatible with the existing container allocation and map task scheduling policies because of different requirements for the format of the cluster topology information. We modified VABM to make it compatible with these policies as well as our extensions to these policies.

E. Support for Heterogeneous clusters

Hadoop clusters are very often built using commodity hardware. Each system might have a different configuration and attached storage devices. Thus, to get the best performance out of the available hardware, heterogeneous clusters (with bare-metal nodes and VMs) are the best option. For example, if some nodes have a large number of cores, then launching VMs on those nodes will most likely give us the best performance. If we have less number of cores on a node, using it directly will most likely give us the best performance. This is because with large number of cores, the physical node might not be able to fully utilize all the available cores. However, when running Hadoop on this heterogeneous cluster, we have to make sure that the topology detection logic and virtualization-aware policies are aware of the heterogeneity of the cluster, which is not the case with default Hadoop.

The policies described above are for virtualized environments. However, there may be times where users may want to run a heterogeneous cluster or even a bare-metal cluster. So, we need to make some changes so that the policies proposed above can be optimally be applied in these cases. For heterogeneous clusters, we used the proposed policies, but extended the topology detection module to ensure that the policies are correctly and optimally applied to these cases. The extended topology detection module selects a unique Host name (the *hostname*) for each bare-metal node. Since the *hostname* is unique, the Host name for each bare-metal node will be unique. Thus, there will be no Host local node for the bare-metal node, which will ensure that policies described above will work correctly for bare-metal nodes. At the same time, the topology detection module will work the same for any VM, and we will be able to fully utilize the benefits of our proposed policies.

IV. Performance Evaluation

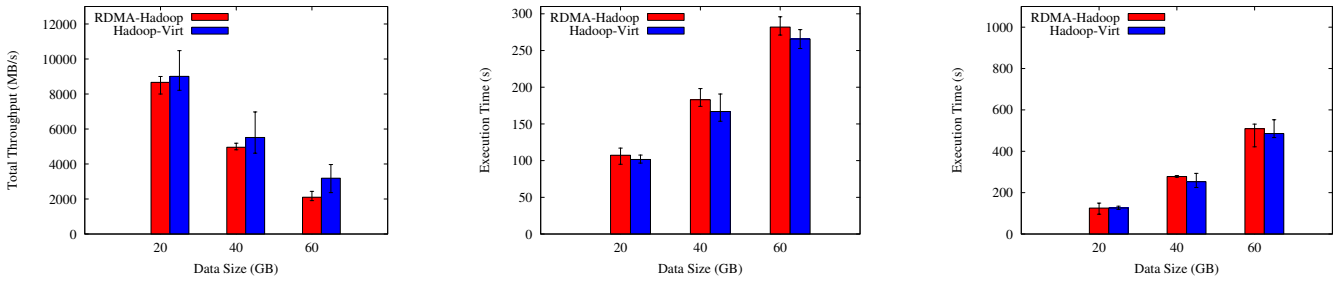
A. Experiment Setup

Our testbed consists of 9 physical nodes on the Chameleon Cloud [30], where each node has a 24-core 2.3 GHz Intel Xeon E5-2670 (Haswell) processor with 128 GB main memory and is equipped with Mellanox ConnectX-3 FDR (56 Gbps) HCAs and PCI Gen3 interfaces. We use CentOS Linux 7.1.1503 (Core) with kernel 3.10.0-229.el7.x86_64. In addition, we use the Mellanox OpenFabrics Enterprise Distribution MLNX_OFED_LINUX-3.0-1.0.1 to provide the InfiniBand interface with SR-IOV support, OpenJDK 1.7.0_91 as the Java package, and KVM as the Virtual Machine Monitor (VMM). For consistency, we use the same OS and software versions for the virtual machines as well.

We have used the standard benchmark suite that comes with Apache Hadoop (v 2.7.1) for our experiments. All benchmarks are run using RDMA-Hadoop 0.9.9 (based on Apache Hadoop 2.7.1) and Hadoop-Virt (based on RDMA-Hadoop 0.9.9). The results have been averaged over three runs to ensure fair comparison.

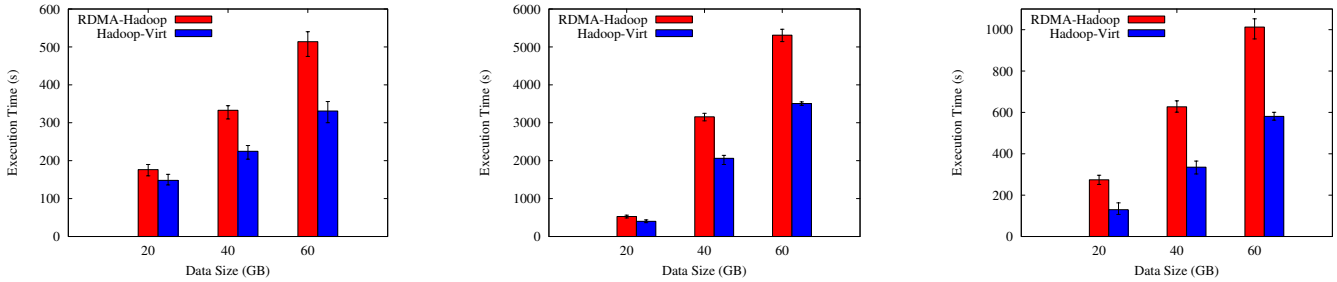
We run 65 VMs in total, with 8 VMs each on 8 bare-metal nodes (totaling 64 VMs) to be used as Hadoop slave nodes, and 1 VM on 1 bare-metal node to be used as the Hadoop master node. 70% of the RAM disk is used for data storage. HDFS block size is kept to 256 MB. The NameNode runs in the master node of the Hadoop cluster and the benchmark is run in the NameNode. Each NodeManager is configured to assign a minimum of 4 GB memory per container. We also make sure that the total number of containers launched per bare-metal node is the same for all cases. This ensures that in all cases the total physical resources used are the same.

Traditionally, Hadoop is deployed with each slave node running 1 NodeManager and 1 DataNode. However, in virtualized environments, it may be beneficial to separate the data (HDFS) and compute (YARN) components of Hadoop, so they can be scaled independently, and any change in one component doesn't affect the other. Thus, for deploying Hadoop, we use 2 modes - Default Mode and Distributed Mode. In the Default Mode, we run Hadoop in the traditional manner, and in the Distributed Mode, we run NodeManagers and DataNodes in separate nodes. So, in our testbed, for the Default Mode we have 64 slaves nodes running 1 NodeManager and 1 DataNode each and for the Distributed Mode, we have 32 slave nodes running 1 NodeManager each and 32 additional slave nodes running 1 DataNode each. The Distributed Mode, thus allows us to scale HDFS and YARN independently allowing for more



(a) TestDFSIO Write Throughput (b) TeraSort Execution Time (c) Sort Execution Time

Figure 5. Performance Comparison of RDMA-Hadoop and Hadoop-Virt for the Default Mode

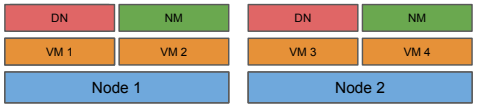


(a) Wordcount Execution Time (b) PageRank Execution Time (c) Sort Execution Time

Figure 6. Performance Comparison of RDMA-Hadoop and Hadoop-Virt for the Distributed Mode

control and flexibility. Figure 7 shows the setup of the two deployment modes for 4 VMs on 2 nodes. We run 96 map tasks and 48 reduce tasks in the Default Mode and 48 map tasks and 24 reduce tasks in the Distributed Mode.

(a) Default Mode



(b) Distributed Mode

Figure 7. Hadoop Deployment Modes

B. Default Mode

For the Default Mode, we tested our proposed design with TestDFSIO Write, TeraSort, and Sort. Figure 5 shows the results of our analysis. We see upto 34%, 8.8%, and 9% improvement with Hadoop-Virt as compared to RDMA-Hadoop for TestDFSIO Write, TeraSort, and Sort, respectively.

C. Distributed Mode

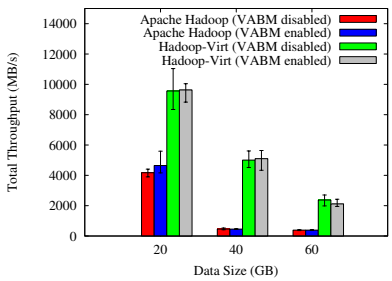
For the Distributed Mode, we ran tests with Wordcount, PageRank, and Sort. Figure 6 shows the results of our analysis. We see upto 35.5%, 34.7%, and 52.6% improvement with Hadoop-Virt as compared to RDMA-Hadoop for Wordcount, PageRank, and Sort, respectively. Overall, we see much more improvement for the Distributed Mode as compared to the Default Mode. This is because in the Default Mode, most of the map tasks and container allocations are Node local. For the Distributed Mode, there are no Node local map task or container allocations since the NodeManagers and DataNodes are running on separate nodes. Thus in this case, RDMA-Hadoop makes all Rack local allocations, whereas Hadoop-Virt makes all Host local allocations, reducing inter-host network traffic and leading to large performance improvements.

D. Virtualization Aware Block Management

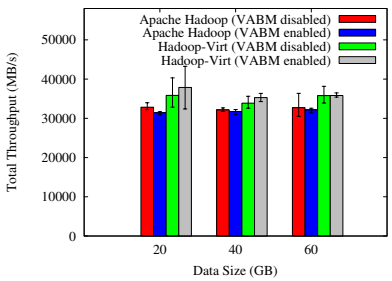
To see the impact of virtualization-aware block management (VABM) on performance and fault tolerance, we ran some experiments with Apache Hadoop and Hadoop-Virt. For measuring fault-tolerance in virtualized environments, we define a new term known as 'Fault Tolerance Level'. The Fault Tolerance Level of a Hadoop cluster is defined as the maximum number of bare-metal node failures that the cluster can handle without data loss. To find the value of this metric, we first generated 60 GB of data using TeraGen, then shut down bare-metal nodes and checked the status of data blocks using the Hadoop *fsck* command. Figure 8 shows the results of this experiment. Firstly, we observe that overall, Hadoop-Virt performs much better than Apache Hadoop. This can be attributed to the RDMA-enabled and virtualization-aware designs in Hadoop-Virt. Secondly, enabling VABM does not have a significant impact on performance. However, it increases the Fault Tolerance Level of the cluster. Finally, we observe that Hadoop-Virt with VABM enabled delivers the best performance as well as Fault Tolerance Level.

E. Application Level Evaluation

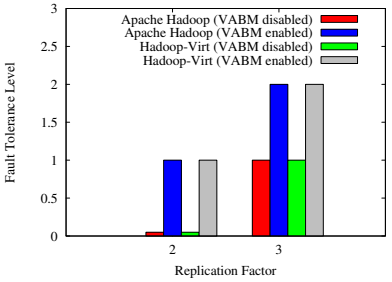
To see whether our proposed designs actually show benefits at the application level, we did some evaluations with two



(a) TestDFSIO Write Throughput

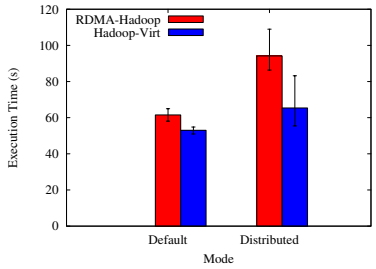


(b) TestDFSIO Read Throughput

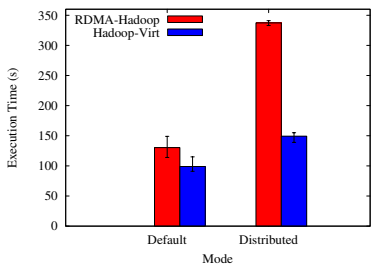


(c) Fault Tolerance Level

Figure 8. Virtualization-aware Block Management (VABM)



(a) CloudBurst Execution Time



(b) Self-join Execution Time

Figure 9. Application Level Evaluation

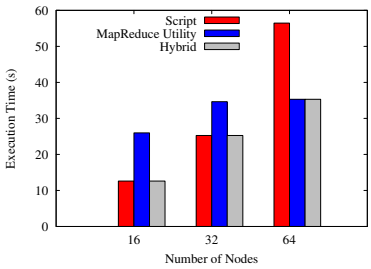


Figure 10. Performance Comparison of topology detection script, MapReduce-based utility, and Hybrid Mode

applications - CloudBurst [31] and Self-join from the PUMA benchmark suite [32]. For CloudBurst, we used the sample data provided with the application, while for Self-join we used 30 GB of data. Figure 9 shows the results of our analysis. We observe 13.8% and 24% improvement in execution time with the Default Mode, and 30.5% and 55.7% improvement with the Distributed Mode for CloudBurst and Self-join, respectively. We see improvement with both modes, which proves the efficiency of our approach.

F. Topology Detection

We proposed two methods to detect the underlying network topology of the Hadoop cluster - topology detection script and MapReduce-based utility. To find which method works better, we ran both methods on our testbed and varied the number of nodes (VMs). We also compared these results with the Hybrid mode of the topology detection module. The result of this analysis is shown in Figure 10. We observe that the script doesn't scale well and the execution time increases linearly. The MapReduce-based utility scales extremely well and the execution time remains more or less stable on increasing the number of nodes. Since the script is run on a single node and is based on ssh, whereas the MapReduce-based utility is built on the scalable and distributed MapReduce framework, these results are expected. That said, we observe that the execution time of the script is smaller as compared to the MapReduce-based utility for 16 and 32 nodes. Thus, for a small sized cluster, the script is a better option, whereas for a large sized cluster, the MapReduce-based utility is a better option. The Hybrid mode thus automatically selects the best out of these two options and delivers the best performance.

V. Related Work

There have been several papers which propose new techniques for improving Hadoop performance in virtualized environments. In this section, we discuss the works most relevant to our paper.

CAM [5] is a topology-aware resource manager to minimize network flow cost for cloud platforms. It uses a flow-network based algorithm to extract best performance for MapReduce under the specified constraints. CAM also provides with a topology server to get information about the cluster topology. However, the implementation of the topology detection process is not described.

Hammoud et al. [33] propose a Locality-Aware Reduce Task Scheduler (LARTS) for enhancing MapReduce performance in Hadoop. LARTS attempts to maximize data local reduce tasks and at the same time prevent scheduling delays, skew and system underutilization.

A white paper by VMware [2] proposes virtualization extensions to Hadoop. In their designs, they propose extensions to task scheduling and replica management in Hadoop. However, their work is based on Hadoop 1.0 and they don't consider detection of topology in their work.

Bu et al. [3] present an approach to reduce interference and increase locality in task scheduling for MapReduce applications in virtual clusters. Their approach includes an adaptive delay scheduling algorithm for improving data locality and an interference aware scheduling policy.

In [7], Nanduri et al. propose an approach for increasing job throughput in the MapReduce framework. They use machine learning to co-locate compatible jobs and maintain resource balance. They show significant improvement when comparing

their approach with the default capacity scheduler in Hadoop.

In [4], Ibrahim et al. present a novel system called CLOUDLET, an implementation of the MapReduce framework optimized for virtualized platforms. Their system emphasizes on reducing the amount of data sent over the network by performing locality-aware operations.

Although all of these papers present topology-aware designs to improve Hadoop performance in virtualized clusters, most of them do not consider topology detection or how it can be automated.

VI. Conclusion and Future Work

In this paper, we proposed a new library called Hadoop-Virt which provides with virtualization and topology aware designs in Hadoop as well as an automatic topology detection module for virtualized clusters. Hadoop-Virt provides virtualization-awareness in all of the four main Hadoop components. Our designs included extensions to the map task scheduling and container allocation policies in Hadoop to improve performance in virtualized environments. Our proposed topology detection module automatically detects the underlying topology of the cluster and provides it to the Hadoop framework. It has multiple modes of operation (static and dynamic) and provides with different methods (Script and MapReduce-based utility) to detect the cluster topology. Our evaluations show that Hadoop-Virt outperforms RDMA-Hadoop in both the Default and Distributed Modes of deployment at the benchmark as well as the application level. Also, with virtualization-aware block management, Hadoop-Virt provides the best level of fault tolerance.

In the future, we plan to make additional improvements to different Hadoop components for better performance in virtualized environments like reducing resource contention between co-located VMs. We also plan to add support for containers.

References

[1] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Locality-aware Resource Allocation for MapReduce in a Cloud," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 58.

[2] VMware, "Hadoop Virtualization Extensions on VMware vSphere 5," *Technical White Paper. VMware, Inc*, 2012.

[3] X. Bu, J. Rao, and C.-z. Xu, "Interference and Locality-Aware Task Scheduling for MapReduce Applications in Virtual Clusters," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*. ACM, 2013, pp. 227–238.

[4] S. Ibrahim, H. Jin, B. Cheng, H. Cao, S. Wu, and L. Qi, "CLOUDLET: Towards MapReduce Implementation on Virtual Machines," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*. ACM, 2009, pp. 65–66.

[5] M. Li, D. Subhraveti, A. R. Butt, A. Khasymski, and P. Sarkar, "CAM: A Topology Aware Minimum Cost Flow Based Resource Manager for MapReduce Applications in the Cloud," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*. ACM, 2012, pp. 211–222.

[6] "Apache Hadoop," http://www.hadoop.apache.org.

[7] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma, "Job Aware Scheduling Algorithm for MapReduce Framework," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 724–729.

[8] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The Rise of Big Data on Cloud Computing: Review and Open Research Issues," *Information Systems*, vol. 47, pp. 98–115, 2015.

[9] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya, "Big Data Computing and Clouds: Trends and Future Directions," *Journal of Parallel and Distributed Computing*, vol. 79, pp. 3–15, 2015.

[10] Y. Yan, C. Chen, and L. Huang, "A Productive Cloud Computing Platform Research for Big Data Analytics," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2015, pp. 499–502.

[11] R. Grossman, "The Case for Cloud Computing," *IT Professional*, vol. 11, no. 2, pp. 23–27, March 2009.

[12] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[13] "PCI-SIG Single-Root I/O Virtualization Specification," http://www.pcisig.com/specifications/iov/.

[14] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High Performance Network Virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471 – 1480, 2012, communication Architectures for Scalable Systems.

[15] J. Liu, "Evaluating Standard-Based Self-Virtualizing Devices: A Performance Study on 10 GBE NICs with SR-IOV Support," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, April 2010, pp. 1–12.

[16] M. Musleh, V. Pai, J. Walters, A. Younge, and S. Crago, "Bridging the Virtualization Performance Gap for HPC Using SR-IOV for InfiniBand," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, June 2014, pp. 627–635.

[17] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor." in *USENIX Annual Technical Conference, General Track*, 2001, pp. 1–14.

[18] T. Agarwal, A. Sharma, A. Laxmikant, and L. V. Kalé, "Topology-aware Task Mapping for Reducing Communication Contention on Large Parallel Machines," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2006, pp. 10–pp.

[19] T. Hoefler and M. Snir, "Generic Topology Mapping Strategies for Large-Scale Parallel Architectures," in *Proceedings of the international conference on Supercomputing*. ACM, 2011, pp. 75–84.

[20] "InfiniBand Trade Association," http://www.infinibandta.com.

[21] "RDMA Hadoop," http://hibd.cse.ohio-state.edu/overview/.

[22] "TOP500 Supercomputing Sites," http://www.top500.org/.

[23] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, "High Performance RDMA-based Design of HDFS over InfiniBand," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 35.

[24] M. Wasi-ur Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D. K. Panda, "High-Performance RDMA-Based Design of Hadoop MapReduce over InfiniBand," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 1908–1917.

[25] X. Lu, N. S. Islam, M. Wasi-ur Rahman, J. Jose, H. Subramoni, H. Wang, and D. K. Panda, "High-Performance Design of Hadoop RPC with RDMA over InfiniBand," in *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013, pp. 641–650.

[26] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE, 2010, pp. 1–10.

[27] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[28] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

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

[30] "Chameleon," http://chameleoncloud.org/.

[31] M. C. Schatz, "CloudBurst: Highly Sensitive Read Mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[32] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "PUMA: Purdue MapReduce Benchmarks Suite," 2012.

[33] M. Hammoud and M. F. Sakr, "Locality-Aware Reduce Task Scheduling for MapReduce," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 570–576.