V-Hadoop: Virtualized Hadoop Using Containers

Srihari Radhakrishnan
University of Waterloo
Email: s2radhak@uwaterloo.ca

Bryan J. Muscedere
University of Waterloo
Email: bmuscede@uwaterloo.ca

Khuzaima Daudjee
University of Waterloo
Email: kdaudjee@uwaterloo.ca

***Abstract*— MapReduce is a popular programming model used to process large amounts of data, and open-source implementations of MapReduce such as Hadoop are generally best suited for large, homogeneous clusters of commodity machines. Many businesses cannot afford to invest in such infrastructure while some are reluctant to use cloud services due to data security and privacy concerns. In this paper, we present V-Hadoop, a framework that leverages Linux containers to allow users to run Hadoop jobs efficiently without requiring large, expensive, physical machine clusters. We describe our design and implementation of V-Hadoop, and show that it can effectively support cluster-level parallelism. We experimentally demonstrate that V-Hadoop is a viable solution that performs competitively compared to solutions designed for large clusters.**

I. Introduction

The MapReduce [1] framework is a specialized programming model for processing large amounts of data at scale across large clusters of commodity machines. Apache Hadoop [2] is a popular implementation of MapReduce and currently supports an ecosystem of tools [3] used widely by researchers and businesses. Since Hadoop's shift to open-source in 2011, MapReduce has become popular for processing large datasets.

Companies like Google, Facebook, and Amazon run MapReduce on datacenter-scale clusters to process hundreds of petabytes of data a day [4]. As other businesses look to deploy Hadoop to meet their data processing needs [2], many of them do not have the need or the capital to purchase and maintain large clusters of physical machines. Since MapReduce is generally utilized with highly parallelized, distributed tasks spread across a homogeneous cluster, small or heterogeneous clusters restrict the benefits of parallelism and result in batch jobs running for undesirable lengths of time. The advent of public cloud services has introduced some degree of flexibility in terms of renting large clusters as opposed to buying them, but with increasing security threats to personal and confidential data, many businesses are reluctant to migrate their data to low-cost commercial cloud-computing datacenters [5].

Though Hadoop is used by many enterprises to run MapReduce jobs on datacenter-scale clusters, it is not well optimized for running large datasets on clusters with a smaller number of nodes [2] as the default 1:1 mapping between Hadoop nodes and physical machines restricts the amount of parallelism.

In view of these challenges, there has been an emphasis on developing better solutions to run distributed jobs while being hardware agnostic. Recent work in this space [6], [7] explores the use of virtualization to provide a unified view of a cluster of physical machines as a single resource. Experiments examining the benefits of virtualizing Hadoop nodes using Virtual Machines (VMs) [8], [9] show that virtualizing Hadoop is promising for the reasons of improved scheduling and resource utilization. Though VMs pave the way for virtualizing Hadoop, they are far from efficient – performance issues with virtualized stacks exist because VMs introduce an unnecessary resource overhead from having to run their own kernel on top of the host operating system.

To address the lack of performant solutions in this space, we present the **V-Hadoop** system which leverages container-based virtualization to address the performance issues associated with deploying Hadoop using VMs. By leveraging container-based virtualization, our V-Hadoop framework is capable of starting and managing multiple Hadoop nodes on and across multiple physical machines where the number of Hadoop nodes is typically greater than the number of physical machines. This decouples the degree of parallelism from the number of physical nodes, effectively increasing parallelism in the system.

V-Hadoop enables a Hadoop cluster to scale to additional nodes when computer resources are overutilized and scale down to fewer nodes during machine underutilization. We demonstrate that V-Hadoop has distinct performance advantages over traditional Hadoop clusters for specific categories of jobs, is generally performance competitive with traditional Hadoop clusters, and maintains this advantage with scale.

II. Background

Also known as operating system-level virtualization, container-based virtualization is an emergent technology that allows users to run multiple operating system instances across a single host. As the popularity of this virtualization method has grown rapidly over the past several years, solutions such as Linux Containers (LXC), Docker and OpenVZ have emerged. LXC is popular due to its ease of use, useful API, and small size [10].

In Linux-based hosts, container-based virtualization software leverages two separate features of the Linux kernel: control groups (cgroups) and namespaces [11]. Namespaces is a Linux kernel feature that places computer resources into an abstraction such that processes in each namespace believe they have exclusive use of that resource. Cgroups is a Linux kernel feature that segregates processes into groups, allowing the OS to allocate and limit the processes given to each group. Both these features form the basis for Linux-based container virtualization [11].

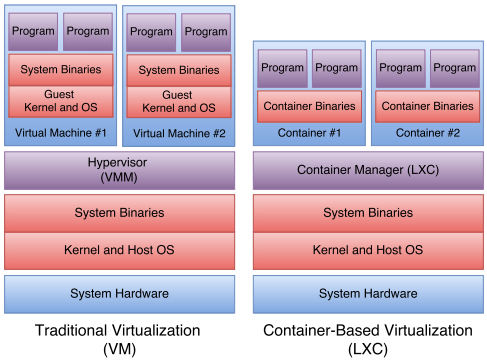

Fig. 1: *The typical program stack for hypervisor-based virtualization (left) compared to container-based virtualization (right).*

While containers appear to be similar to VMs that use hypervisor-based virtualization, there are many significant differences. Figure 1 shows the high-level difference between hypervisor-based virtualization and container-based virtualization. First, in a hypervisor, since each VM is guaranteed a fixed amount of physical resources set by the user, resources allocated to a VM can be underutilized. Comparatively, containers avoid this problem by sharing physical resources with their host machine, taking up machine resources only when required. Additionally, since VMs emulate full machines, they contain an entire operating system. A recent study [12] comparing the boot times of Linux containers and VMs showed that containers boot six times faster than VMs. These differences between containers and VMs make containers a more attractive option for efficiently running processes in isolation without having to manually set resource limits.

III. V-Hadoop System

The V-Hadoop framework must carry out three tasks: *a*) cluster elasticity; *b*) container setup; and *c*) container management. Each of these operations is important in providing our high-level goals of elasticity, hardware agnosticism and management of a Hadoop cluster. The remainder of this section describes V-Hadoop in the context of these three tasks.

A. Cluster Elasticity

Similar to how Hadoop Distributed File System (HDFS) and Yarn [13] manage data and MapReduce tasks across Hadoop nodes, V-Hadoop dynamically manages the containerized Hadoop nodes to optimize physical resources and provide real time elasticity of the virtual cluster without manual intervention. V-Hadoop is organized into a master-worker architecture where one physical machine is denoted the *master* and the rest of the machines in the cluster are denoted *workers*. To easily maintain a running V-Hadoop cluster in the presence of machine failures, the master should run on the same machine that has the container running the master Hadoop node. This would help quickly resolve any container crashes of that master node. Traditionally, the master node is protected by a failover mechanism such as Zookeeper [13].

In our framework, the purpose of this dichotomy is important; the master machine manages the placement of containers on the system and makes decisions regarding scaling the cluster up or down. The workers tasks carry out decisions made by the master node and are responsible for starting and stopping containers and running Hadoop configuration tasks. Further, the workers continually check the state of their respective systems for container crashes or resource utilization problems. If a worker detects a problem, it contacts the master for a decision. The master is responsible for making a decision since it has a global view of the cluster; it can take the states of all workers into account before deciding where containers need to be spawned and/or shut down.

B. Container Setup

V-Hadoop initially starts all of the containers, making copies of containers where necessary and configures Hadoop on each cluster. This phase is triggered by the master machine and propagates to other machines over a period of time. When the master is first initialized, it detects all the physical machines in the cluster and configures the worker processes on each machine.

Since this phase can experience long periods without any communication between master and worker, we make use of heartbeats to check the status of the physical machines in the cluster. Once deployed, the master continually sends a HeartbeatRequest message to each worker. The workers respond with a HeartbeatResponse message. In the event that the master does not receive a reply from a worker within a set interval, it assumes the node is dead. Since heartbeat messages have to be sent out continuously, the messages are small. The master sends these HeartbeatRequest messages on a fixed interval, TPROBE, which can be set programmatically. The payload and time-to-live (TTL) of the Heartbeat messages are small enough that even frequent messages between the master and worker do not cause significant network traffic.

Once the master has determined which workers are running, it initiates the setup of the V-Hadoop cluster. Based on the user's initial requirements for the number of nodes in the Hadoop cluster, the master starts and configures virtual Hadoop nodes on the local machine and instructs the workers to do the same on their machines. Since V-Hadoop is specifically concerned with managing a physical cluster consisting of a small number of high performance machines, we fill up the machines with containers greedily, starting with the master machine and proceeding to do the same on worker machines in order of lowest round-trip time (RTT) first. The logic for this placement is handled by the container management aspect of our framework and is discussed in greater detail in Section III-C.

Importantly, in addition to creating and starting new containers each master and worker configures the container to become a Hadoop node by modifying the necessary configuration parameters to setup HDFS and to connect the containers to form a working Hadoop cluster. The NameNode and Re-

sourceManager of the Hadoop cluster will reside on the master machine.

When new nodes need to be started on a worker, the master issues a network request to that worker with information on the number of nodes required. The desired worker then creates the required number of containers and responds to the master with a SUCCESS message and payload consisting of a list of the currently running nodes with their network addresses. Many techniques can be applied to reduce delays at the master as it waits for responses from workers while creating multiple nodes, e.g., asynchronous network requests, non-blocking network I/O, callback mechanisms. The details of how this can be achieved are implementation specific. The creation and replication of containers is implementation specific as well, and depends on the type of container technology being used. This is discussed in more detail in section III-D. Once the master has received a list of running Hadoop nodes from all the workers, it then notifies the user that Hadoop can now be used. Since HDFS and Yarn are unaware of the underlying node-container abstraction, the code managing the namenode, datanodes and nodemanager remains unchanged from Hadoop's fully-distributed mode. This natural abstraction offered by virtualization could be used to streamline several features in Hadoop such as the distributed file system.

C. Container Management

Our framework defines a container management mode that occurs once a MapReduce job is running on a V-Hadoop cluster. The primary purpose of this mode is to periodically scan the physical machines in the V-Hadoop cluster and check for underutilization, overutilization and failed containers so that containers can be appropriately added and/or removed.

To carry this out, the worker on each machine periodically checks machine health by aggregating machine statistics and comparing it against a pre-determined ceiling for CPU, memory and disk usage. Each worker also checks the health of the containers on the system with the same periodicity. If a worker detects that a machine is underutilized or overutilized or that a container has crashed, it sends a DecisionRequest message to the master indicating that a container management decision needs to be made. DecisionRequest messages queue up in temporal order in a queue on the master. The master picks up DecisionRequests from the DecisionQueue and processes them one by one. Each DecisionRequest triggers messages sent to all the workers allowing the master to obtain a global view of the utilization of all machines and the health of all containers. With this global snapshot of the physical cluster and the list of containers running on each machine, the master balances the system load by shutting down containers in overutilized machines and spawning new ones in underutilized ones. DecisionRequests triggered by crashed containers are also dealt with in a similar fashion.

The V-Hadoop framework ensures fair resource utilization across the physical machines in the cluster and automatically scales up or scales down the size of the V-Hadoop cluster based on the state of currently running jobs.

D. Implementation

We implemented a prototype called the V-Hadoop Container Manager (VCM) which implements the framework described in Section III and integrates a container management system. Overall, the VCM's tasks are: *a*) start and stop containers on a system; *b*) configure Hadoop on all containers running across all physical machines to form a cluster; *c*) pull resource information from each physical machine in the V-Hadoop cluster; and *d*) perform simple container management including adding and removing containers on-demand.

Due to the V-Hadoop framework's flexibility, the VCM can be run on a single machine or across a small cluster of physical machines. To allow for this, VCM is comprised of two processes: *MainVCM* and *VCMLite*. These processes form a master-worker architecture where the MainVCM process coordinates the cluster.

Based on this, if running the VCM on a single machine, it will run in *local mode* with the MainVCM process running by itself. In *distributed mode*, where multiple machines are running the VCM, one machine will run the MainVCM process whereas the rest run VCMLite. In this mode, the physical machine running MainVCM is denoted as the master and the other machines are the workers.

IV. Evaluation

In this section, we compare the performance of MapReduce jobs on V-Hadoop with the two standard Hadoop modes, fully-distributed and standalone (pseudo-distributed).

A. Test Environment

Our evaluation environment consisted of a 10-machine cluster. Each machine contained twelve Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz cores, 256GB RAM and 2TB of disk storage split across 3 physical disks. Our tests used three widely recognized Hadoop benchmarks: TestDFSIO, MRBench and TeraSort. These were specifically chosen to compare the system performance of V-Hadoop against the two Hadoop modes with respect to different resources: TestDFSIO for disk intensive jobs, MRBench for CPU intensive jobs and TeraSort for CPU, disk and network intensive jobs. The machines in the fully-distributed setup were connected using a single TopOfRack(ToR) Switch on an Intel I350 Gigabit Network Connection. For standalone mode, these tests were run on a single machine. V-Hadoop tests were conducted on a single machine running a virtual cluster of 10 containers. The base container in V-Hadoop was configured with Ubuntu 14.04 and Hadoop version *2.7.1*. We configured HDFS to map to the container's file system and set the replication factor to 1, since replicating blocks on the same physical machine is redundant. Further, replication on the same disk could slow down jobs due to disk contention.

B. TestDFSIO Benchmark

TestDFSIO is a benchmark that stresses the Hadoop Distributed File System(HDFS) and is split into read and write tests. It is useful to stress test HDFS as it measures the cluster's

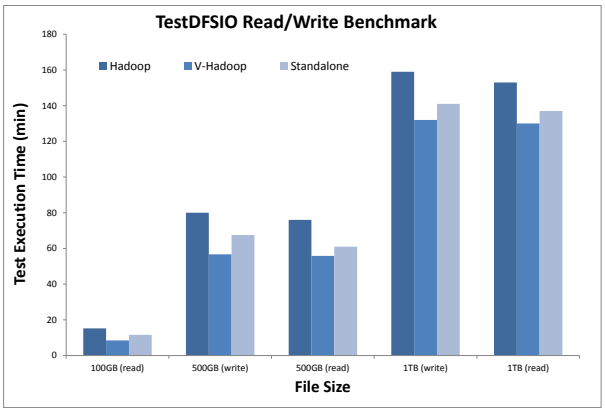

Fig. 2: *TestDFSIO read/write benchmarks execution time for different Hadoop modes (10GB file size).*

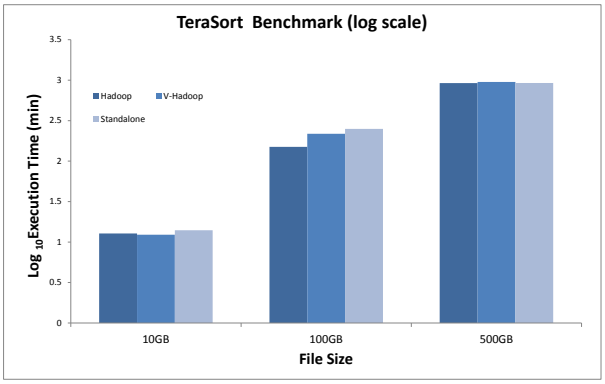

Fig. 3: *TeraSort benchmark results for the three Hadoop modes (log scale execution time).*

storage throughput and performance in terms of both disk and network I/O. The TestDFSIO benchmark is designed to use one map task per file. Splits are defined such that each map operation gets only one filename to read or write from. We ran TestDFSIO by varying the number of files and file sizes, and measuring the benchmark execution time of writes and reads. Fig 2 shows a plot of execution time versus the number of files for a ten-node cluster. In comparing the TestDFSIO benchmark throughput for the three modes, we find that V-Hadoop and standalone Hadoop perform better for writes and reads compared to the fully-distributed mode, primarily because V-Hadoop and standalone modes obviate the need for network I/O. We observe that network I/O in the fully-distributed mode is significant enough to cause a marked difference in execution times, even though the fully-distributed mode is writing and reading files from 10 machines, thereby having much lower disk and memory contention than V-Hadoop and standalone modes. Fig 2 shows that V-Hadoop performs better than Hadoop for I/O-intensive operations, and this performance difference is maintained as the size of the dataset is scaled up.

C. TeraSort Benchmark

The TeraSort benchmark is used to test the HDFS and Yarn layers of the Hadoop cluster. It is a popular benchmark often used in practice and by industry to measure the standard of a Hadoop cluster. The TeraSort benchmark is a good measure of how well the Hadoop cluster is configured in terms of the number of Map/Reduce tasks, and how well they are balanced compared to the number of disks, cores and machines in the cluster. The goal of TeraSort is to sort a large number of 100-byte records as quickly as possible. A full TeraSort benchmark run consists of the following three steps: *a*) Generating the input data using TeraGen, *b*) Running TeraSort on the input data, and *c*) Validating the sorted output data via TeraValidate. Through these three steps, the benchmark performs considerable amount computation and I/O and is considered to be representative of real MapReduce programs. Terasort overrides the specified replication factor so only one copy is written to HDFS. TeraValidate reads the sorted data and verifies whether it is in order using one map task per file and combines the results using a single reduce task to check if the files are contiguous.

We ran the three stages of TeraSort on data sets ranging from 1GB to 0.5TB on the three Hadoop modes, for increasing block sizes. Fig 3 shows a comparison of the TeraSort benchmark runtimes averaged over all three phases (TeraGen, TeraSort and TeraValidate) for the three modes. The benchmark was run on datasets of sizes 0.01TB, 0.1TB and 0.5TB with block size 512MB on a logarithmic scale of execution time (minutes). We observed that for smaller file sizes, the difference in execution speeds was negligible. Even for larger file sizes, V-Hadoop performed comparably to the fully-distributed mode facing heavier disk contention compared to fully-distributed Hadoop.

D. MRBench Benchmark

MRBench is a benchmark designed to iterate a short MapReduce program a number of times. MRBench checks whether small job runs are responsive and running efficiently on the cluster. MRBench is complementary to the Terasort benchmark, which tests V-Hadoop's MapReduce performance on large files. We chose MRBench as a way to compare the MapReduce layer of V-Hadoop to that of a fully-distributed Hadoop cluster. The MRBench tests were performed on a minute long MapReduce job which was looped over 50, 500, 1000 and 5000 times. Fig 4 shows a comparison of V-Hadoop, standalone and fully-distributed Hadoop for each of these runs. We observe that the execution times between V-Hadoop and Hadoop remain comparable as the number of runs is scaled up from 50 to 5000 even though V-Hadoop runs on a single machine, as opposed to the fully-distributed cluster running on 10 machines. Comparing execution times of V-Hadoop and standalone mode, we see that V-Hadoop consistently performs better, and the gap between the execution times continues to widen as the number of runs are scaled up. This experiment illustrates why container based virtualization is a clear winner for performance reasons alone – we can achieve results comparable to a fully-distributed cluster on a physical cluster which is an order of magnitude smaller in size.

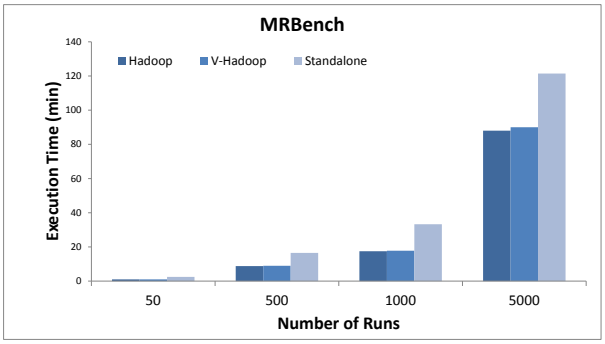

Fig. 4: *MRBench benchmark results for the three Hadoop modes.*

V-Hadoop provides the best of both worlds by combining the resource utilization of the standalone mode with the scalability of the fully-distributed mode. It utilizes machine resources more efficiently by design while still providing the flexibility of a fully-distributed Hadoop cluster as new nodes can span physical machines by leveraging containers. Our experiments show that even though fully-distributed Hadoop performs slightly better on CPU intensive benchmarks than V-Hadoop, the performance gain is not significant enough to justify the use of more than one physical machine. Additionally, while V-Hadoop and standalone Hadoop run on identical hardware and have comparable performance in the TeraSort benchmark, V-Hadoop is the better choice mainly because of the distinct advantage it offers in highly parallel jobs (such as MRBench) and I/O heavy jobs (as shown in the TestDFSIO benchmark experiments). Furthermore, V-Hadoop provides a layer of abstraction atop a physical cluster that allows users to easily scale-up or scale out their hardware as well as expand their Hadoop cluster while being agnostic to the underlying hardware.

V. Related Work

[9], [14] examined the feasibility of running a virtual Hadoop cluster using a collection of virtual machines on a physical machine. VMWare has looked at the performance of running a virtual Hadoop cluster using proprietary virtualization technology [8]. While their study demonstrates that running multiple Hadoop instances on a single machine can improve throughput of MapReduce jobs, virtualizing using VMs can have significant overhead. Recent work [15] has evaluated virtualized Hadoop's performance but have found that the performance of I/O-intensive jobs is more sensitive to the virtualization overhead than that of CPU-intensive jobs due to shared I/O. Apache Mesos is a cluster management tool that provides users with the ability to share clusters between distributed computing frameworks such as Hadoop [7]. While Mesos is different from V-Hadoop in specific functionality, Mesos takes advantage of containerization to provide resource isolation between distributed computing frameworks and applications [7]. Despite this, Mesos does not run multiple containers of the same framework on a single worker machine.

VI. Conclusion

In this paper, we presented V-Hadoop, a framework that allows users to run a virtual Hadoop cluster across any number of machines while being agnostic to the underlying hardware. The prototype version of our V-Hadoop framework can manage a V-Hadoop cluster across multiple physical machines, and can perform simple resource-based scheduling of Linux containers across physical machines by adding or removing containers in the cluster dynamically based on resource usage and availability. Our experimental evaluation shows that V-Hadoop performs comparably to a fully-distributed Hadoop cluster. V-Hadoop allows for elastic clusters that can utilize the resources of the underlying physical infrastructure, providing significant management and cost benefits to the user.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[2] T. White, *Hadoop: The Definitive Guide*, ser. Oreilly and Associate Series. O'Reilly, 2012. [Online]. Available: https://books.google.ca/books?id=drbI_aro20oC

[3] J. R. et al. The hadoop ecosystem table. [Online]. Available: https://hadoopecosystemtable.github.io/

[4] R. Bohn and J. Short, "How Much Information? 2010 Report on Enterprise Server Information," Dec. 2011. [Online]. Available: http://hmi.ucsd.edu/howmuchinfo_research_report_consum_2010.php

[5] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security & Privacy*, no. 6, pp. 24–31, 2010.

[6] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, no. 3, pp. 81–84, 2014.

[7] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," University of California, Berkeley, Tech. Rep., 2010.

[8] J. Buell, "A benchmarking case study of virtualized hadoop performance on vmware vsphere 5," VMWare, Tech. Rep., 2011.

[9] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, "Evaluating mapreduce on virtual machines: The hadoop case," in *Cloud Computing*. Springer, 2009, pp. 519–528.

[10] D. Lezcano. (2015, oct) Lxc api documentation. [Online]. Available: https://linuxcontainers.org/lxc/documentation/

[11] M. Cohen and C. Pereira, "Cisco application-centric infrastructure (aci) and linux containers," Cisco, Tech. Rep., 2014.

[12] K.-T. Seo, H. Hwang, I. Moon, O. Kwon, and B. Kim, "Performance comparison analysis of linux container and virtual machine for building cloud," 2014.

[13] A. S. Foundation. (2015, jul) Apache hadoop 2.4.1 documentation. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html

[14] M. Gomes Xavier, M. Veiga Neves, and C. Fonticielha de Rose, "A performance comparison of container-based virtualization systems for mapreduce clusters," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, Feb 2014, pp. 299–306.

[15] M. Ishii, J. Han, and H. Makino, "Design and performance evaluation for hadoop clusters on virtualized environment," in *Information Networking (ICOIN), 2013 International Conference on*. IEEE, 2013, pp. 244–249.