Experiences in Using OS-level Virtualization for Block I/O

Dan Huang[1], Jun Wang[1], Qing Liu[2], Jiangling Yin[1], Xuhong Zhang[1], Xunchao Chen[1]
[1]Department of Electrical Engineering and Computer Science
[1]University of Central Florida, Orlando, FL
[2]Oak Ridge National Lab, Oak Ridge, TN
[1]{ dhuang, jwang, jyin, xzhang, xchen}@eecs.ucf.edu [2]{liuq}@ornl.gov

ABSTRACT

Today, HPC clusters commonly use Resource Management Systems such as PBS and TORQUE to share physical resources. These systems enable resources to be shared by assigning nodes to users exclusively in non-overlapping time slots. With virtualization technology, users can run their applications on the same node with low mutual interference. However, the overhead introduced by the virtual machine monitor or hypervisor is too high to be accepted, because efficiency is key to many HPC applications. OS-level virtualization (such as Linux Containers) offers a lightweight virtualization layer, which promises a near-native performance and is adopted by some BigData resource sharing platforms such as Mesos. Nevertheless, OS-level virtualization's overhead and isolation on block devices have not been completely evaluated, especially when applied to a shared distributed/parallel file system (D/PFS) such as HDFS or Lustre. In this paper, we thoroughly evaluate the overhead and isolation involved in sharing block I/O via OS-level virtualization on the local disk and D/PFSs. Meanwhile, to assign D/PFS storage resources to users, a middleware system is proposed and implemented to bridge the configuration gap between virtual clusters and remote D/PFSs.

1. INTRODUCTION

In current HPC clusters, Resource Management Systems (RMS), such as PBS/TORQUE [8] are used to schedule and allocate resources. Facilitated by PBS/TORQUE, HPC cluster resources can be shared among multiple users. Typically, PBS and TORQUE accept users' jobs by executing batches and then reserving physical nodes in a cluster to run those jobs. However, without state-of-the-art virtualization technology to isolate assigned resources, applications from different users will work with mutual interference on the same node.

Recently, cloud computing has been widely deployed in data centers and privately owned clusters because it can provide high efficiency and elastic resource consolidation [19]. Cloud computing platform is becoming an infrastructure for various MPI-based HPC Analytics and data-intensive analytics such as MapReduce [12], Hadoop [21]. For example, Amazon EC2 has provided a dedicated cluster equipped with high-end virtual nodes with high-speed network connection for scientists to run simulations and data analytics. In such environment, physical machines are virtualized, and virtual machines (VMs) are bridged as virtual clusters. Users run data-intensive and HPC analytics on such virtual clusters.

Nevertheless, despite its benefits, cloud computing and virtualization technologies have not been fully accepted by HPC communities because of the overhead introduced by the hypervisor. Several studies have been conducted to evaluate the performance overhead of virtualization. In general, researches [15, 20, 22] have demonstrated that traditional hypervisor-based virtualization (such as Xen [10], VMware [9] and KVM [2]) has a high performance overhead, specially in terms of memory and I/O (up to 40%).

Compared to hypervisor-based Virtualization, OS-level virtualization (such as Docker [1], OpenVZ [6] and Linux Containers (LXC) [3]) implements a lightweight virtualization layer in Linux kernel, which promises a low performance overhead [20, 22] (less than 2% in CPU and Memory). In this setting, OS-level virtualization can be a viable option for HPC data intensive analytics and provide low-overhead performance isolation. The use of OS-level virtualization could improve resource sharing and maintain multiple isolated userspace instances. For instance, Mesos [14] is a platform that uses LXC for sharing a cluster between multiple diverse cluster computing frameworks, such as Hadoop and MPI.

However, the performance of allocating and isolating block device resources via OS-level virtualization hasn't been completely evaluated. This is due to the fact that some functionalities used to control block I/O are still under development and not clearly defined. For instance, Mesos claims its current release only supports assignment of CPU and memory to jobs and will cover block devices in the future. On the other hand, various storage or computing platforms such as Hadoop File System, Map/Reduce, Spark, or MPI can be deployed on current BigData/HPC analysis clusters. Most HPC MPI-based and BigData Hadoop-based jobs retrieve data from remote distributed files systems (DFS) such as HDFS [11] or Lustre [4]. Even if users could isolate their jobs in virtual nodes (in this paper we refer it as **VNode**) and assign local block device resources to them via various virtualization technologies, these jobs may contend with each other over elements within the D/PFS, such as I/O node.

PDSW2015, November 15-20, 2015, Austin, TX, USA
©2015 ACM. ISBN 978-1-4503-4008-3/15/11 ...$15.00
DOI: http://dx.doi.org/10.1145/2834976.2834982.

For instance, when a user runs ParaView [7] to load and visualize scientific data from a remote D/PFS such as Lustre, Lustre can simultaneously stage data for other jobs, such as the resilient checkpoint jobs issured by Scalable Checkpoint/Restart (SCR). As a result, the interference on the shared remote D/PFS will result in an unpredictable performance.

In this paper, we thoroughly evaluate the overhead and isolation of OS-level virtualization's block I/O control. Based on the evaluation results, we analyze the limitations and advantages of block I/O control. For allocating and isolating a D/PFS's I/O resources, we design a I/O Middleware system, which enables administrators to assign not only local block I/O to users but also remote D/PFS resources dynamically. The results show that with this middleware system, the high-priority applications' quality of service can be guaranteed by assigning as much of a D/PFS's I/O resources as they request.

2. BACKGROUND FOR OS-LEVEL VIRTUALIZATION

In recent years, the emergence of OS-level virtualization has become an alternative to hypervisor based VMs. Hypervisor based virtualization provides an abstraction layer on top of the hardware or host operating system. However, OS-level virtualization as a Linux kernel module takes effect at the kernel level, offering abstractions directly for a group of processes. The OS-level virtualization shares the same operating system kernel, as a result, it will be less isolated than hypervisor based virtualization. In OS-level virtualization, the isolation and resource allocation in OS-level virtualization is done by Linux Namespace and Control Groups (cgroups). Linux Namespace creates a set of namespaces such as PID and Network Namespaces, which allow each virtual node to have a private process space and a network protocol stack. CGroups controls the resource usage per process group. Specifically, cgroups is used to limit or prioritize CPU, memory and I/O usage for OS-level virtualization. The block I/O control within cgroups provides two functionalities, proportional weight division and I/O throttle. The proportional weight is implemented in the Completely Fair Queuing (CFQ) I/O scheduler. This policy allows users to assign weights to specific process groups. I/O throttle is used to set an upper limit on the read/write bandwidth of a specific block device.

3. I/O MIDDLEWARE SYSTEM DESIGN

In the D/PFS setting, when multiple users share a cluster via OS-level virtualization, CPU and memory resources can be configured into each virtual cluster. However, controlling I/O resources that are shared on a D/PFS for a specific user remains an unsolved challenge in current resource control platforms. Also, the high priority I/O intensive jobs may interfere with other jobs on a shared D/PFS. For instance, when a user performs interactive analytics such as ParaView, other users are simultaneously reading logs from the same D/PFS. In order to guarantee that ParaView will finish in time, we need to throttle the I/O rate of other low-priority users' reads to a lower level so that more bandwidth can be allocated to ParaView. In this section, we propose a middleware system, VNode Sync, to achieve this functionality in D/PFS (HDFS or Lustre).

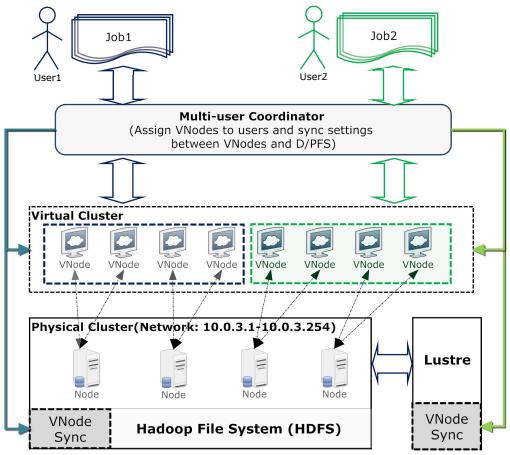


Figure 1: The Architecture of VNode Cluster

3.1 System Architecture

As shown in Figure 1, a physical cluster is a set of physical nodes connected by switches. If physical machines are attached with local disks, the Hadoop File System (HDFS) could be deployed on the physical cluster. In addition, if scientists intend to migrate data from remote storage nodes, remote parallel file systems (Lustre) can be accessed as well. VNodes are process containers in physical nodes and can be assigned resources such as CPU, memory, block I/O etc via cgroups. A virtual cluster is formed by a group of VNodes and assigned to users. VNode Sync is designed for implementing block I/O configurations on D/PFSs. Partitioning I/O bandwidth from a shared D/PFS is currently an unsolved problem. We will introduce this component in the following subsection in detail. On top of virtual cluster, there is a component named Multi-user Coordinator, which distributes shared D/PFS resources to users based on their priority.

3.2 Synchronizing Users' Disk I/O on D/PFS

CGroups, a module in OS-level virtualization, can throttle or weight the local disk I/O for different process groups. Linux Container, Mesos and Docker have implemented their resource controlling features via cgroups. However, in most BigData/HPC applications, such as MPI-based ParaVivew, Hadoop-based MapReduce jobs, retrieve data from remote D/PFSs (Lustre or HDFS). In such situations, the I/O configuration on local disk I/O is not feasible to control a remote D/PFS's I/O. In other words, users are assigned a number of VNodes to run their jobs. The CPU and memory resources are assigned and isolated in VNodes. The shared I/O resources of a D/PFS can not be controlled via current resource allocation mechanisms since the I/O configurations on users' VNodes can not take effect on a remote D/PFS. In order to bridge this gap, we propose a component called VNode Sync, which is based on the block I/O throttle functionality of cgroups.

Typically, each user is assigned two sets of VNodes. The first one is used to construct virtual clusters to run jobs. The second set of VNodes is deployed in the D/PFS's storage nodes and each VNode is used for controlling the I/O bandwidth of the host storage node based on the corresponding user's priority. The VNode Sync accepts users' I/O configurations from the Multi-user Coordinator and applies them on a D/PFS's storage nodes via Linux cgroups. On each storage

node, a number of VNodes are initiated for D/PFS users. Technically, VNode Sync intercepts users' I/O requests and the D/PFS's I/O request handlers and then places these I/O handlers into the users' configured VNodes to control the I/O throughput to specific users. We take HDFS as an example to specify this mechanism. In HDFS, when a user requests to read a file, the master node will tell the user to retrieve file chunks from a storage node. Then, the user constructs a TCP connection with the storage node to request specific file chunks. When the requests are accepted, this storage node will spawn a number of threads to handle the TCP connection, load data from the disk, and transfer data. At this point, our proposed VNode Sync will intercept both the user's requests and the storage node's request handler processes, and put these processes into the user's configured VNode. Inside VNode, cgroups will apply configured I/O policies, such as the proportion of shared block I/O resources.

The Multi-user Coordinator is deployed on a physical node to partition the shared D/PFS's I/O resources to users based on their priority. In order to assign the aggregate I/O resources of D/PFSs to users efficiently, we design Algorithm 1, which is used to throttle the block I/O of active users based on their priorities. For example, the total bandwidth of each storage node is 200 MB/s. If there are three users, whose priorities are 4, 8 and 8, according the algorithm, the first user's block I/O bandwidth will be throttled to under 40 MB/s on each storage node and other two users will both be throttled to under 80 MB/s.

Algorithm 1 Priority-based Algorithm for Throttling D/PFS I/O

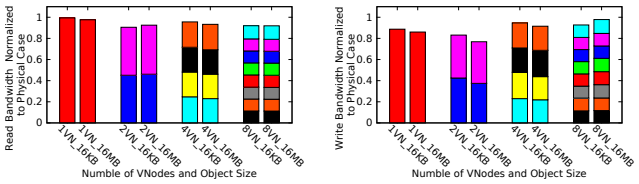
1: Let N be the number of users
2: Let $P = \{p_1, p_2, ..., p_n\}$ be the set of N priority values for N users. Priority range is from 1 to 100 and 100 is the highest.
3: Let BW be the total aggregate I/O bandwidth that D/PFS's storage nodes can provide.
4: **Steps:**
5: // throttle block I/O at tht_i to the $User_i$ based on the priorities
6: $i = 1$
7: **if** $N = 1$ **then**
8: $tht_1 = BW$
9: **else**
10: **for** i from 1 to N **do**
11: $tht_i = BW \cdot \frac{p_i}{\sum_{j=1}^{n} p_j}$
12: set block I/O throttle rate tht_i to $user_i$'s VNode
13: **end for**
14: **end if**

4. VNODE EVALUATION ON SINGLE NODE TESTBED

In this section, we evaluate the overhead and isolation involved with sharing block I/O via VNodes on a single node testbed whose configuration is shown in Table 1. We run a set of control experiments on block devices (hard disk) with an HPC I/O benchmark called MPI-IO Test [5], which was developed by the Los Alamos National Lab. Before each test, the block I/O buffer, Page Cache, is cleaned to guarantee that the requested data is accessed from block devices instead of the buffer.

Table 1: The Configuration of Single Node Testbed

Make& Model	Dell XPS 8700
CPU	Intel i7 Processor, 64 bit, 18 MB L2, 2.8 GHz, 4 cores
RAM	8 × 2 GB
Internal HD	1x Western Digital SATA 7200rpm 2 TB
Internal SSD	Intel SRT SSD 32 GB
Local File System	EXT3
Operating System	CentOS 6 64-bit, kernel 2.6.32_504.8.1.el6



Read Overhead Write Overhead

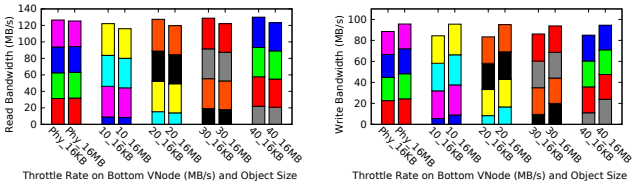
Figure 2: The read and write overhead evaluation: The x-axis represents the number (1, 2, 4, 8) of the benchmark processes running in physical nodes or VNodes, as well as object size (16 KB, 16 MB). The y-axis is the aggregate I/O bandwidth normalized to physical case.

4.1 Read and Write Overhead in VNode

In the experiment, we run MPI-IO benchmark instances on the single node testbed with and without affiliation to VNodes. Both testcases spawn the same number of concurrent processes (1, 2, 4, and 8), each of which writes or reads 2 GB of data to the hard disk with medium and large object sizes (16 KB and 16 MB)(N-to-N write). In Vnode testcase, each process is affilated to a Vnode. In Figure 2, we can see that the write overhead VNode is less than 25%, and as the concurrency increases to 8 processes, the overhead decreases to 3% (object size: 16 MB) and 5% (object size: 16 KB). The worst read overhead is less that 10%, which is lower what we anticipated.

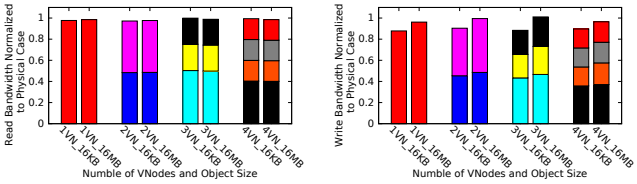
4.2 The Isolation Evaluation of Throttle Functionality

In this test, we evaluate the isolation of throttle functionality by running 4 concurrent processes on 4 VNodes (one process affiliated to one VNode). We throttle one VNode's disk I/O bandwidth to different thresholds. We consider iso-



Read Throttle Isolation Write Throttle Isolation

Figure 3: The read and write throttle isolation evaluation: Each stacked bar represents 4 benchmark processes that are performed in 4 VNodes.The x-axis represents the throttle rate (10, 20, 30, 40 MB/s) to the first VNode, as well as object size (16 KB, 16 MB). "Phy" represents all concurrent processes that are run on the physical node without VNode affiliation. The y-axis represents the aggregate read/write bandwidth.



Read Weight Isolation Write Weight Isolation

Figure 4: The read and write weight isolation evaluation: The x-axis represents the number of VNodes (1, 2, 3, 4) for each test case, as well as access object size (16 KB, 16 MB). For different numbers of VNodes, the weight distributions are (1:100%), (2: 50%, 50%), (3: 50%, 25%, 25%), (4: 40%, 20%, 20%, 20%). The y-axis is the bandwidth of the process in VNode normalized to the aggregate bandwidth of the physical testcase.

lation to be the amount of mutual interference among the peer VNodes. The results, as shown in Figure 3, show that on the hard disk, throttle effects could be largely influenced by other concurrent processes that are not being throttled. This is due to the fact that throttled processes are not isolated from other processes, they are only limited to the assigned bandwidth but not guaranteed to be assigned that bandwidth.

We can further explain the isolation of throttle more from the kernel level. The throttle module places a method, *cgroup_io_throttle*, at the point where every I/O request is initiated. This method is used to determine whether the I/O request submitted by a process is accepted or delayed by the I/O controller. In each time slot, the I/O controller can accept at most B bytes ($B = Throttle_{rate} * time$) of I/O requests for each VNode. If the I/O quota for the current time slice is used, the *cgroup_io_throttle* will simply cause the submitted process to sleep in order to slow down the I/O. The advantage of the throttle module is its simplicity. On the down side, operating at the I/O request creation level implies that throttle I/O may also interfere with the I/O priority policies implemented at the I/O scheduler level. For example, if a block device is set to adopt CFQ as the I/O scheduler, the I/O requests from the VNode's processes are first throttled by the *cgroup_io_throttle()*, and then will wait to be scheduled with the other I/O requests by the CFQ scheduler. Because of this, in the throttle isolation test, the process in each VNode is influenced by the processes in the other VNodes (throttle rate is 40 MB/sec but the process in VNode is only assigned 20.7 MB/sec).

4.3 The Isolation Evaluation of Weight Functionality

In this experiment, we evaluate the performance of weight functionality by running 4 concurrent processes on 4 VNodes (one process affiliated to one VNode). Each process reads/writes 2 GB of data from a file (N-to-N) and each VNode is set at different weights. The comparison physical testcase is running 4 concurrent processes on the sinle node testbed without VNode affiliation. The bandwidth of each process in VNode testcase is normalized to the aggregate bandwidth of the physical testcase. In Figure 4, the results show that this CFQ-based weight functionality is as accurate as the kernel file specifies and the overhead of the weight function is less that 8%. As opposed to the throttle function, the weight module does not suffer from mutual interference and can provide effective isolation.

At the kernel level, the weight module of OS-level virtualization is implemented within the CFQ scheduler. In traditional CFQ, each process maintains a CFQ Queue (CFQQ) to store the waiting I/O requests and allocates time slots to access the block device for each of the queues. On the contrary, the weight module creates a CFQ Queue Group (CFQQG) for each VNode. The length of the time slot for each CFQQG is determined by the weight rate configured in each VNode. As a result, the weight functionality will not suffer the same isolation problem as the throttle functionality.

5. EVALUATION EXPERIMENTS ON DISTRIBUTED AND PARALLEL FILE SYSTEMS

In Section 3, we proposed an I/O Middleware to enable D/PFS I/O resources to be shared with multiple users via OS-level virtualization. In this section, we evaluate the performance of our I/O Middleware on D/PFSs. All of the following experiments are conducted on PRObE's Marmot Cluster [13], where we reserve 17 nodes. For the tests on HDFS, one node is configured as the master node, the other 16 nodes are configured as storage nodes. We run benchmark instances and real HPC applications on these 16 storage nodes. To test Lustre, one node is configured as the master node, 8 nodes are configured as storage nodes and the other 8 nodes are client nodes, used to run the benchmark application. Before each test, every storage node's buffer area, Page Cache, is cleaned to guarantee that the requested data is read from block devices.

5.1 Evaluating I/O Middleware on HDFS and Lustre

In this experiment, each storage node is configured with a VNode, which contains the D/PFS instance (HDFS or Lustre). The I/O Middleware is deployed on the storage nodes and the client nodes of HDFS and Lustre to apply users' block I/O throttle rates to the VNodes. HDFS is configured normally with 3-way replication and the size of a chunk file is set to 64 MB. When reading data from HDFS, the MPI-IO Test benchmark spawns 16 processes on 16 storage nodes (one process per node). The process will attempt to read from a local disk if the chunk file exists locally. This is referred to as *With Data Co-locality*. If the required chunk file is not on the local disk, the client will randomly read data from the remote storage node that stores the chunk file, referred to as *Without Data Co-locality*. For the tests on Lustre, the MPI-IO Test benchmark spawns 8 processes on 8 client nodes, each of which hosts 1 process. Each process reads a 2 GB file from Lustre. In the cluster bar chart (Figure 5), the first red bar represents the result of the test *With Data Co-locality*; the second green bar is the test *Without Data Co-locality*; the third blue bar is the N-to-N access test on Lustre and the forth pink bar represents the N-to-1 access on Lustre. The x-axis represents the block I/O throttle rates on the VNodes that host the D/PFS instance and the benchmark processes. "W/O_VN" represents the D/PFS instances that are run on the physical machine without VNode affiliation. From the bar charts, we can see that our I/O Middleware can effectively control the aggregate band-

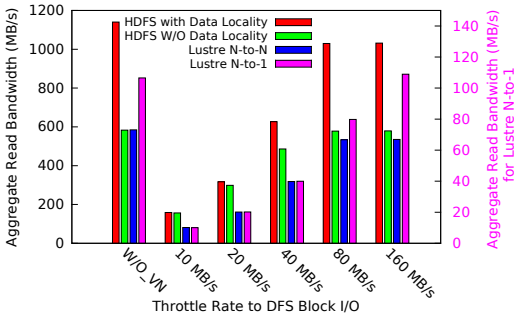


Figure 5: Evaluate I/O Middleware: MPI-IO benchmark runs on HDFS (16 storage nodes and 16 client processes) and Lustre (8 storage nodes and 8 client processes). The x-axis represents the throttle rate on each storage node via I/O Middleware. The left y-axis represents the aggregate bandwidth. The right y-axis represents the aggregate bandwidth of Lustre N-to-1 read.

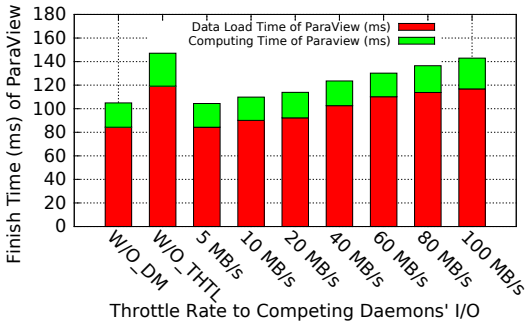


Figure 6: Evaluate I/O Middleware via Real Applications: PareView instance and two background daemons retrieve data from HDFS simultaneously. PareView competes with the daemons for the shared HDFS's I/O. The x-axis represents the I/O bandwidth throttled on the daemons. The y-axis represents the finish time of ParaView.

width of D/PFSs and introduces negligible overhead. This is due to the fact that a D/PFS's network and communication latency also introduces certain amounts of performance overheads. On the contrary, the overhead introduced by VNode's block I/O control accounts for a small fraction of the overall overhead.

5.2 Evaluating I/O Middleware via Real Applications

In this evaluation, we suppose that there are multiple users running multiple I/O intensive jobs on the same D/PFS (HDFS as an example in this case), one of the jobs, such as ParaView, has a high priority. ParaView is an MPI-based interactive HPC visualization tool. Users desire low-latency response once a visualization request is sent to ParaView's render server. The render server loads data from the D/PFS and then renders and outputs the results into a file. In this test, each of the 16 storage nodes are configured with two VNodes. One VNode contains a ParaView process, and the other VNode hosts two background daemons that read data from HDFS. In such settings, we attempt to use the I/O Middleware to throttle the background daemons so as to guarantee that the high priority job (ParaView) will run first. In Figure 6, we plot the finish times of ParaView while increasing the throttle rate to background daemons via the I/O Middleware. The x-axis represents throttle rates to background daemons. "W/O_DM" in the x-axis represents the case of running ParaView exclusively without any background daemons. "W/O_THTL" represents the case of running ParaView and background daemons without throttling the block I/O of daemons. When we set the block I/O rate of the background daemons to 5 MB/s, ParaView's finish time is close to the finish time of the "W/O_DM" case. As the throttle rate to background daemons increasing, the finish time of ParaView is also increasing (up to 140% of the "W/O_DM" case).

6. RELATED WORK

Using virtualization technology to consolidate and share the cluster resources to multiple users is becoming more popular in both the HPC and BigData communities. ACIC [17], proposed by Liu, is an automatic I/O configurator for optimizing HPC applications' I/O on the cloud platform. Niu [18] proposed a computing model for dynamically reserving and resizing resources on the cloud environment to minimize costs.

In addition, many researchers in different research domains have performed a series of investigations on OS-level virtualization from evaluations to applications. Authors [15, 20, 22], have evaluated the overhead (CPU, memory and disk) of OS-level virtualization compared with the traditional hypervisor based VM. The results show that OS-level virtualization outperforms hypervisor largely on memory and disk throughputs. However, the authors do not completely evaluate the overhead and isolation of the block I/O control. Multilanes [16] is a virtual storage system for OS-level virtualization. It builds an isolated I/O stack for each virtual node to eliminate contentions on shared kernel data structures and locks, which are the performance bottlenecks, while applying OS-level virtualization to fast block devices (e.g. SSD). There are some researchers in UC Berkeley who proposed and implemented a platform, Mesos [14], for sharing commodity clusters with multiple users and multiple computing platforms such as Hadoop and MPI. Mesos takes advantage of OS-level virtualization (LXC) to provide cluster resource sharing (only CPU and memory) in a fine-grained manner. However, it does not provide any mechanism to share a cluster's block device resources with users and computing platforms.

7. CONCLUSION

In this paper, we investigate the overhead and isolation performance of OS-level virtualization's block I/O control on a single node and D/PFS. In most cases, the block I/O control of OS-level virtualization introduces less than 10% overhead. The weight functionality, which is implemented at CFQ, introduces at most 8% overhead and shows good performance isolation. The throttle functionality of block I/O control also introduces low performance overhead but has limited performance on isolation and its result will be largely influenced by peer processes. In addition, throttle functionality is used to implement the I/O Middleware to share the I/O bandwidth of D/PFSs to multiple users based on their priorities. It performs effectively with negligible overhead on D/PFSs and real applications.

8. ACKNOWLEDGEMENTS

This work is supported in part by the US National Science Foundation Grant CCF-1527249, CCF-1337244 and National Science Foundation Early Career Award 0953946. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

The experiments of this work are conducted at the PRObE Marmot cluster, which is supported by the National Science Foundation under the following NSF program: Parallel Reconfigurable Observational Environment for Data Intensive Super-Computing and High Performance Computing (PRObE).

9. REFERENCES

[1] Docker, https://www.docker.com/.

[2] KVM. `http://www.linux-kvm.org/page/Main_Page`.

[3] Linux container, https://linuxcontainers.org/.

[4] Lustre filesystem. http://www.lustre.org/.

[5] MPI-IO Test. `http://institute.lanl.gov/data/software/`.

[6] Openvz, http://www.openvz.org.

[7] Paraview, http://www.paraview.org/.

[8] TORQUE Resource Manager `http://www.adaptivecomputing.com/products/open-source/torque/`.

[9] Vmware, http://www.vmware.com/.

[10] Xen, http://www.xenproject.org.

[11] Dhruba Borthaku. The Hadoop Distributed File System: Architecture and Design.

[12] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[13] Garth Gibson, Gary Grider, Andree Jacobson, and Wyatt Lloyd. Probe: A thousand-node experimental cluster for computer systems research. *USENIX; login*, 38(3), 2013.

[14] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: a platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[15] Nikolaus Huber, Marcel von Quast, Michael Hauck, and Samuel Kounev. Evaluating and modeling virtualization performance overhead for cloud environments. In *CLOSER*, pages 563–573, 2011.

[16] Junbin Kang, Benlong Zhang, Tianyu Wo, Chunming Hu, and Jinpeng Huai. Multilanes: providing virtualized storage for os-level virtualization on many cores. In *FAST*, pages 317–329, 2014.

[17] Mingliang Liu, Ye Jin, Jidong Zhai, Yan Zhai, Qianqian Shi, Xiaosong Ma, and Wenguang Chen. Acic: automatic cloud i/o configurator for hpc applications. In *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pages 1–12. IEEE, 2013.

[18] Shuangcheng Niu, Jidong Zhai, Xiaosong Ma, Xiongchao Tang, and Wenguang Chen. Cost-effective cloud hpc resource provisioning by building semi-elastic virtual clusters. In *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pages 1–12. IEEE, 2013.

[19] Jongse Park, Daewoo Lee, Bokyeong Kim, Jaehyuk Huh, and Seungryoul Maeng. Locality-aware dynamic vm reconfiguration on mapreduce clouds. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pages 27–36. ACM, 2012.

[20] Stephen Soltesz, Herbert Pötzl, Marc E Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 275–287. ACM, 2007.

[21] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, original edition, June 2009, ISBN: 0596521979.

[22] Miguel G Xavier, Marcelo Veiga Neves, Fabio D Rossi, Tiago C Ferreto, Timoteo Lange, and Cesar AF De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 233–240. IEEE, 2013.