vLocality: Revisiting Data Locality for MapReduce in Virtualized Clouds

Xiaoqiang Ma, Xiaoyi Fan, Jiangchuan Liu, Hongbo Jiang, and Kai Peng

Abstract

Recent years have witnessed a surge of new generation applications involving big data. The de facto framework for big data processing, MapReduce, has been increasingly embraced by both academic and industrial users. Data locality seeks to co-locate computation with data, which effectively reduces remote data access and improves MapReduce's performance in physical machine clusters. State-of-the-art public clouds heavily rely on virtualization to enable resource sharing and scaling for massive users, however. In this article, through real-world experiments, we show strong evidence that the conventional notion of data locality is unfortunately not always beneficial for MapReduce in a virtualized environment. The observations suggest that the measure of node-local must be extended to distinguish physical and virtual entities. We develop *vLocality*, a comprehensive and practical solution for data locality in virtualized environments. It incorporates a novel storage architecture that efficiently mitigates the shared disk contention, and an enhanced task scheduling algorithm that prioritizes co-located VMs. We have implemented a prototype of vLocality based on Hadoop 1.2.1, and have validated its effectiveness on a typical virtualized cloud platform consisting of 22 nodes. Our experimental results demonstrate that vLocality can improve the job finish time to around a quarter of that for typical Hadoop benchmark applications.

Introduction

Recent years have witnessed a surge of new generation applications involving big data. A typical big data life cycle consists of multiple stages [1]. First, the source data are generated from various devices at different locations and collected via wired/wireless access networks. Second, the collected data are aggregated and delivered through the global Internet to remote data centers for further processing and analysis. Third, the results may be delivered back to end devices to be further exploited. Given the large volume of data to be transmitted and processed, each stage has specific requirements on data transmissions, bringing new challenges to the underlying networking architecture and services.

As an example, MapReduce [2] has become the de facto framework for big data processing, with the ability to harness the power of thousands of interconnected servers in data centers to deal with terabyte and even petabyte data. Such practical implementations as the open source Apache Hadoop have increasingly been embraced by both academic and industrial users.[1] However, not all MapReduce users or potential users have dedicated MapReduce clusters due to various reasons, such as the costly upfront investment in hardware/software and the lack of expertise on their cumbersome configuration, not to mention the challenges in expanding the cluster when the application scale escalates. Fortunately, the readily available clouds provide an alternative solution for big data analytics. Cloud users can rent machines from public cloud providers, say Amazon Web Services (AWS),[2] and deploy the Hadoop stack as well as other standard/customized tools. Thus, they can enjoy the convenient and flexible *pay-as-you-go* billing option, as well as on-demand resource scaling. For example, Yelp, a famous business rating and review site, successfully saved US$55,000 in upfront hardware costs by using MapReduce on AWS, processing 3 TB of data per day.[3]

A MapReduce cluster generally consists of a *master*, which acts as a central controller, and a number of *slaves*, which store data and conduct user-defined computation tasks in a distributed fashion [2]. In a physical machine cluster, each slave node has a *DataNode* that stores a portion of data, and a *TaskTracker* that accepts and schedules tasks. The *NameNode* on the master node hosts the directory tree of all files on DataNodes, and keeps track of the locations of files. A typical MapReduce workflow consists of two major phases, as shown in Fig. 1. First, the map processes (mappers) on slaves read the input data from the distributed file system and transform the input data to a list of intermediate key-value pairs (known as the map phase); the reduce processes (reducers) then merge the intermediate values for the distinct keys, which are stored in the local disks of slaves, to form the final results that are written back to the distributed file system (known as the reduce phase). This distributed paradigm of MapReduce inevitably incurs a large amount of network traffic within/across MapReduce clusters.

Since fetching data from remote servers across multiple network switches can be costly (particularly in clusters/data centers with high overprovisioning ratio), in traditional MapReduce clusters, *data locality*, which seeks to co-locate computation with data, can largely avoid the costly massive data exchange across switches, thereby significantly improving the job finish time of most tasks [3–5]. As one of the most important technical foundations of modern clouds, virtualization techniques (e.g., Xen, KVM, and VMware) allow multiple virtual machines (VMs) to run on a sin-

Xiaoqiang Ma, Hongbo Jiang, and Kai Peng are with Huazhong University of Science and Technology.

Xiaoyi Fan and Jiangchuan Liu are with Simon Fraser University.

This work was supported in part by the National Natural Science Foundation of China under Grants 61572219 and 61502192, a Canada NSERC Discovery Grant, and a Canada NSERC Strategic Project Grant.

[1] Apache Hadoop. http://hadoop.apache.org

[2] Amazon Web Services. http://aws.amazon.com

[3] AWS Case Study: Yelp. http://aws.amazon.com/solutions/case-studies/yelp/

Digital Object Identifier: 10.1109/MNET.2016.1500133NM

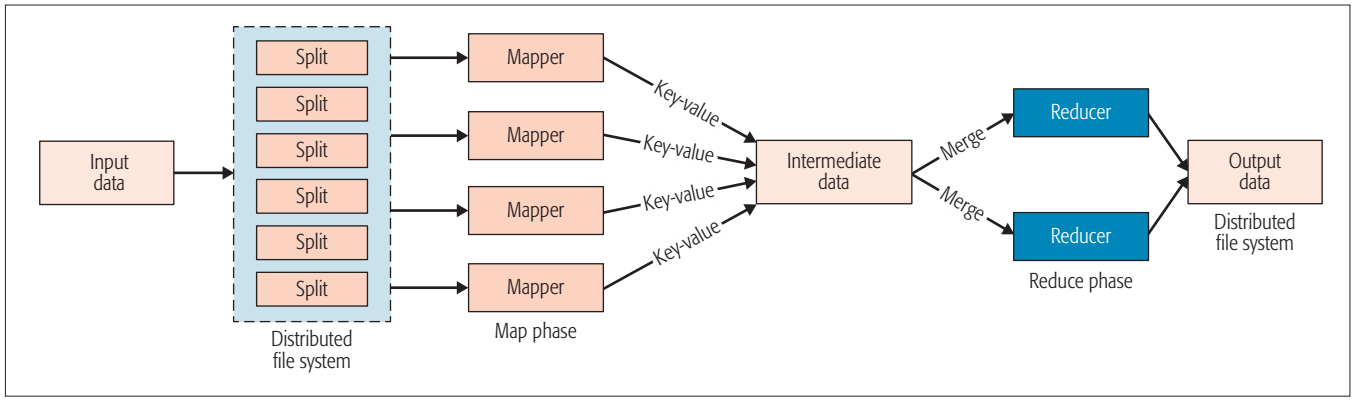

FIGURE 1. A typical MapReduce workflow.

gle physical machine (PM), which achieves highly efficient hardware resource multiplexing and effectively reduces the operating costs of cloud providers. It has been identified that MapReduce jobs running on VMs have significantly longer finish times compared to directly running on their physical counterparts due to such unique characteristics of VMs as resource sharing/contention and VM scheduling [6–8], which is also confirmed by our real-world experiments.

Our experiments suggest that the conventional notion of data locality designed for PMs needs substantial revision to accurately reflect the data locality in virtualized environments. In particular, *node-local*, which indicates that running tasks fetch data in a vicinity, should be extended. Simply distributing data to nearby VMs (i.e., *VM-local*) is not necessarily helpful; only if the VMs are co-located in the same PM (i.e., *PM-local*) will a large portion of congested disk I/O be effectively offloaded through highly efficient memory sharing. Modifying the storage architecture to improve PM-locality, however, is nontrivial, as the current task scheduler in Hadoop is unable to distinguish the difference: when scheduling tasks, co-located VMs have the same priority as the VMs on other PMs in the same rack. To this end, we develop an enhanced task scheduling algorithm that prioritizes co-located VMs. These efforts together lead to the development of *vLocality*, a comprehensive and practical solution toward data locality in virtualized environments. We examine the design issues of vLocality and implement it in Hadoop 1.2.1. Its effectiveness has been validated on a typical virtualized cloud platform, which shows that vLocality improves the job finish time to around a quarter of that for typical MapReduce applications compared to baselines.

The rest of this article is organized as follows. We present our motivational experiments and summarize the key observations. We discuss the design principles of vLocality. We compare vLocality to the default Hadoop system with state-of-the-art data placement [4, 9] in real-world experiments. We provide some further discussion and conclude this article

When Data Locality Meets Virtualization

Real-world MapReduce systems, such as Hadoop and Google's MapReduce, attempt to achieve better data locality through replicating each file block on three servers so that two of them are within the same rack and the remaining one is in a different rack. More advanced data locality solutions have also been developed [4, 5, 10], although mostly working for PM clusters.

Similarly, in a VM cluster, each VM serving as a slave also has a DataNode and a TaskTracker by default. This balanced architecture is very straightforward, and is supposed to provide the best performance since each VM can access data locally, achieving the maximum data locality. On the other hand, a single DataNote can be set up on only one VM to serve all the other co-located VMs. Intuitively, this imbalanced architecture incurs more remote accesses and has a lower degree of data locality, since the VMs without DataNodes need to fetch data remotely.

We have conducted a series of experiments in a testbed cloud platform to understand and compare the performance of data locality under the different configurations above. Our experimental results reveal the distinct characteristics when data locality meets virtualization, which indeed contradict our intuition, suggesting that the conventional data locality strategies working for PMs should be revised.

Our testbed consists of three state-of-the-art Dell servers (OPTIPLEX 7010), each equipped with an Intel Core i7-3770 3.4 GHz quad core CPU, 8 GB 1333 MHz DDR3 RAM, a 1 TB 7200 RPM hard drive, and a 1 Gb/s Ethernet network interface card (NIC). Hyper-threading is enabled for the CPU so that each CPU core can support two threads. All the PMs are interconnected through a gigabit switch. This cluster of controlled scale allows the machines to be interconnected with maximum speed and enables us to closely examine the interplay among all of them, without concern for the background interference from many other machines. We use a widely adopted open source virtualization tool, Xen [11], in which a Xen hypervisor provides the resource mapping between the virtual hardware and the underlying real hardware of the PM. A privileged VM, *Domain0*, is created at boot time and is allowed to use the control interface. The hypervisor works together with the *host OS* running on Domain0 to provide system management utilities. The OS running on an unprivileged domain (*DomainU*) VM is called the *guest OS*, and can only access the resources that are allocated by the hypervisor. The DomainU VMs cannot directly access the I/O devices; rather, the Domain0 VM handles all of the I/O processing. Xen uses the shared memory mechanism for data transfer between co-located VMs [11].

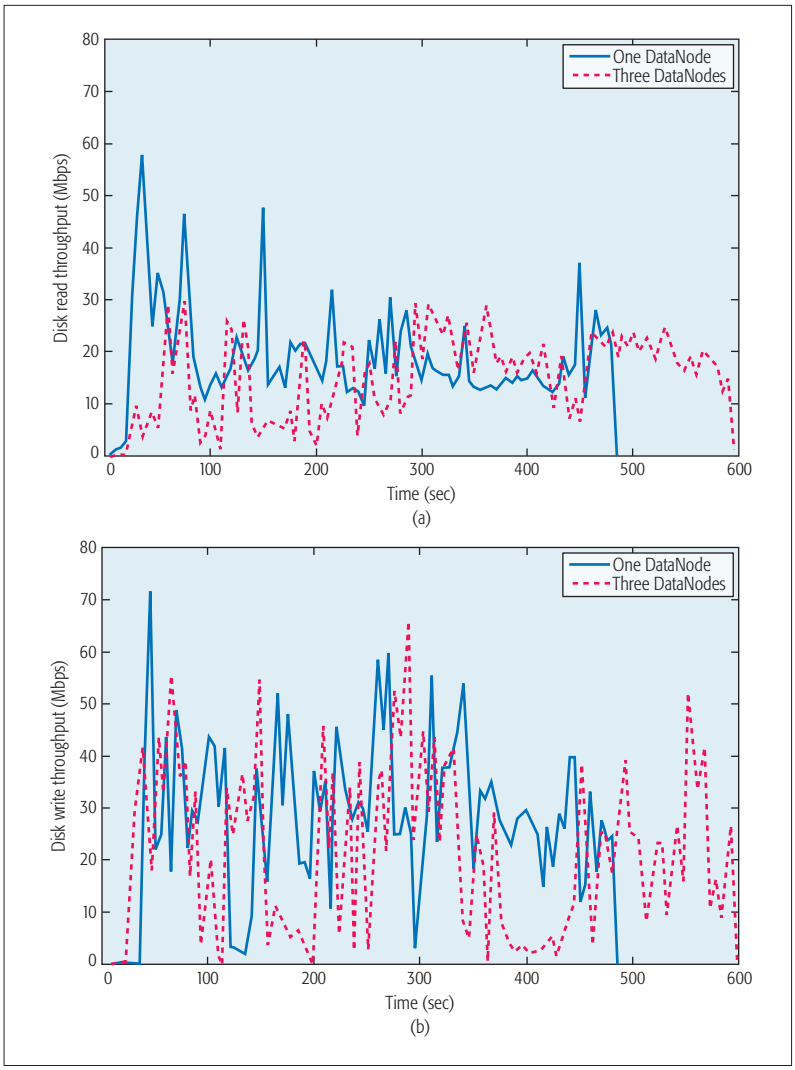

FIGURE 2. Total disk read/write throughput during MapReduce running time.

We use a separate PM as the master node to ensure fast response time with minimized resource contention, and accordingly enable a fair comparison with fully non-virtualized systems. On each of the other PMs, besides the Domain0 VM, we configure three identical DomainU VMs (two virtual CPUs and 2 GB RAM for each) as slave nodes. We use the logical volume management (LVM) system, which is convenient for resizing, to allocate each DomainU VM 100 GB disk space by default. We use the popular Ubuntu 12.04 LTS 64-bit as the operating system, and run Hadoop 1.2.1 on the VMs. Our testbed is configured similarly to public cloud,[4] as well as the setting in [16], with high-end Intel multicore CPUs, Linux-based operating systems, and the Xen virtualization tool. Hence, the observations in our testbed experiments can be reproducible in typical virtualized clouds.

DataNode Placement: Less Is Better

Our first observation is that the number of DataNodes per PM has a remarkable impact on MapReduce's performance. We start from a simple cluster, in which two PMs are interconnected through the switch, one serving as the master node and the other hosting three VMs. The three VMs act as slaves, each having a TaskTracker. We extract 3.5 GB Wikipedia data from the Wikimedia database,[5] and select the widely used `Sort` application as our benchmark, which arranges the lines of text in the input files in alphabetical order. For the DataNode placement, we examine three representative configurations:

1. Setting up a DataNode on only one VM.
2. Randomly selecting two VMs and setting up a DataNode on each.
3. Setting a DataNode on each VM. When there are more than one DataNode, the input data will be almost evenly stored on them.

We run the benchmark application five times for each configuration, and the average job finish times of the above three configurations are 458.0, 487.0, and 524.5 s, respectively. It can be seen that the lower the number of DataNodes per PM, the shorter the job finish time. This contradicts the observation with traditional PM MapReduce clusters. In a PM cluster, the highest data locality will be achieved when all tasks can access the input data from local nodes (*node local*), and the job finish time is generally shorter than that with a lower degree of data locality. However, for virtualized clusters, our results indicate that achieving complete VM locality can be harmful.

A closer look shows that the reasons are threefold. First, many MapReduce tasks, say `Sort` in the example, are both computation and I/O intensive; considering the contention for the shared resources, more DataNodes will significantly increase the burden of VMs and the extent of inter-VM interference. Second, both reading data from and writing data to the Hadoop distributed file system (HDFS) involve meta-data exchange with the master node; having multiple DataNodes also increases the overhead of such information exchange. Third, besides HDFS involved I/O operations, each VM also generates massive intermediate data, which is generally several times more than the amount of input and output data; the intermediate data needs to be written to and read from the local disk of each VM to bridge the map and reduce phases. But the concurrent I/O operations of HDFS and local disks will aggravate the intra- and inter-VM contention for the disk.

The observations suggest that selecting one VM hosting the DataNode to serve all co-located VMs can be a better choice. This architecture overcomes the above drawbacks of one DataNode per VM, and thus mitigates the contention for disk. To understand this, we use the `iotop` tool, which is available in most Linux distros, to measure the real-time disk read/write throughput during the process of two experiments (one DataNode per VM and one DataNode per PM), and plot the total disk read/write throughput of the three VMs in Fig. 2 We can see that the one-DataNode-per-PM setting has a noticeably higher average disk read/write throughput (read: 19.03 MB/s vs. 15.19 MB/s, write: 27.39 MB/s vs. 21.23 MB/s) and lower variation.[6] Further, the data exchange between co-located VMs is very efficient, which has been validated using the `iperf` tool. The measurement results show that the network bandwidth between two co-located VMs exceeds 15 Gb/s, indicating that co-located VMs directly use the memory bus for data exchange.

[4] For example, a Rackspace *General1-2* instance has 2 virtual CPUs and 2 GB RAM.

[5] Available at http://dumps.wikimedia.org/

[6] It is worth noting that `iotop` itself introduces some overhead and thus the job finish time becomes slightly longer, which however exists for all the cases and thus will not affect the relative differences.

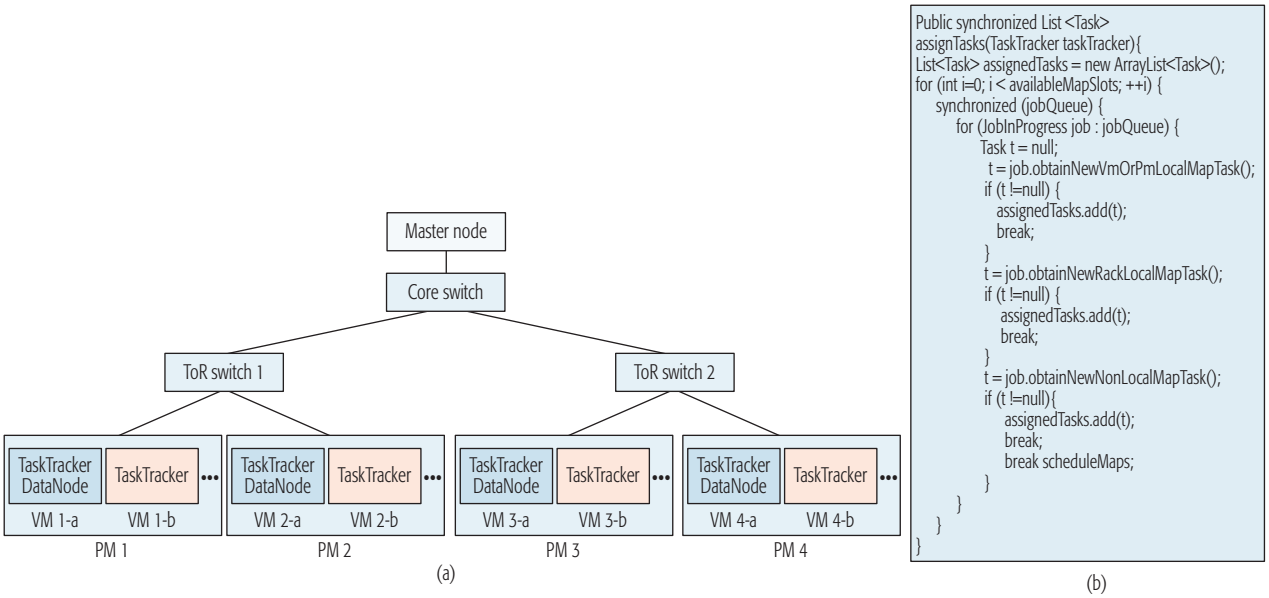

FIGURE 3. a) vLocality architecture design: a core switch is a high-capacity switch interconnecting top-of-rack (ToR) switches, which have relatively low capacity; b) task scheduler in vLocality.

Hence, the added latency caused by remote access across co-located VMs is negligible.

Virtual Locality: Nearest Is Not the Best

To further verify that remote data access across co-located VMs does not add noticeable performance penalty, our other set of experiments use two PMs, which are connected through a switch. For the non-master PM, we have two configurations: in configuration a, we boot up only one DomainU VM, which has both TaskTracker and DataNode; in configuration b, we boot up two DomainU VMs: one has a TaskTracker only, and the other has a DataNode only. Other experimental settings are the same as in the previous experiment. We again run each experiment five times for each configuration. The average job finish times of configurations a and b are 799.0 and 433.3 s, respectively. The result is very interesting. In configuration a, the VM can access all the data from its local disk, while in configuration b, all the input/output data needs to be transmitted between the two VMs. However, the latter is much more efficient in terms of MapReduce job finish time, which not only verifies our previous conjecture that remote access across co-located VMs does not add noticeable performance penalty, but also indicates that MapReduce can significantly benefit from decoupling TaskTracker and DataNode.

vLocality: Architecture Design and Prototype Implementation

Given the observations above, it is necessary to revisit the notation of data locality for MapReduce in virtualized clouds. This inspires our design and development of vLocality, which seeks to improve the effectiveness of locality in the virtualized environment, but with minimized modifications to existing MapReduce implementations. Figure 3a illustrates the architecture of vLocality. For the VMs co-located on the same PM, we only set up a single DataNode on one of them; each other VM on this PM has only TaskTracker.

To reduce the cross-server network traffic during job execution, the task scheduler on the master node usually places a task onto the slave, on which the required input data is available if possible. However, this is not always successful since the slave nodes that have the input data may not have free slots at that time. Recall that the default replication factor is three in the Hadoop systems, which means that each file block is stored on three servers, two of them within the same rack and the remaining one in a different rack. Hence, depending on the distance between DataNode and TaskTracker, the default Hadoop defines three levels of data locality: *node-local*, *rack-local*, and *off-switch*. Node-local is the optimal case when the task is scheduled on the node having data. Rack-local represents a suboptimal case when the task is scheduled on a node different from the node having data, but the two nodes are within the same rack. Rack-local will incur cross-server traffic. Off-switch is the worst case, when all the node-local and rack-local nodes are busy, and thus the task needs to retrieve data from a node in a different rack, incurring cross-rack traffic. When scheduling a task, the task scheduler uses a priority-based strategy: node-local has the highest priority, whereas off-switch has the lowest.

In virtualized MapReduce clusters, however, the three-level design is not enough to accurately reflect the data locality. For example, in Fig. 3a, for a file block stored on VM 1-a, scheduling the task on VM 1-b or VM 2-b is considered identical by the task scheduler since both are rack-local. However, data exchanges between co-located VMs (e.g., VM 1-a and VM 1-b) are much faster than those between remote VMs (e.g., VM 1-b and VM 2-b), and hence the two cases should have different scheduling priorities. To this end,

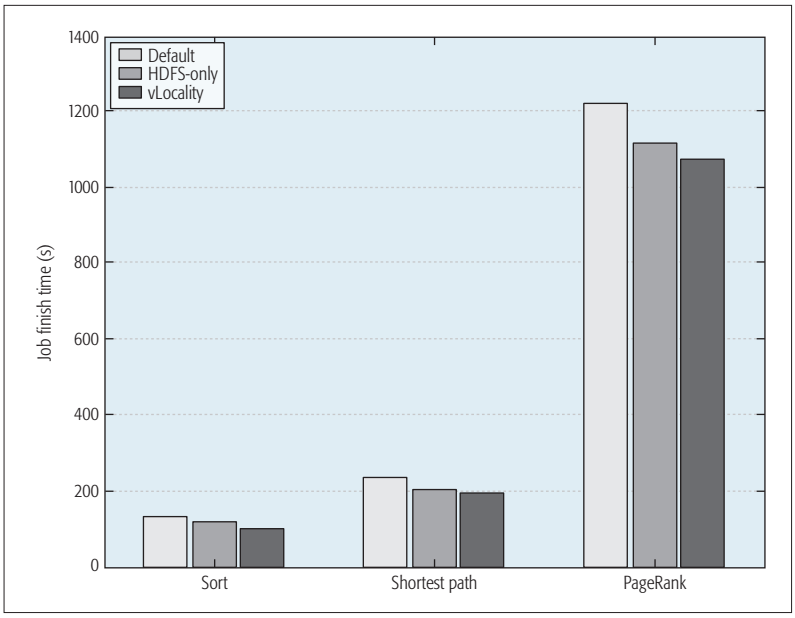

FIGURE 4. Comparison of job finish time.

we modify the original three-level priority scheduling strategy by splitting node-local into two priority levels, *VM-local* and *PM-local*, in virtualized MapReduce clusters, defined as follows:

- **VM-local**: A task and its required data are on the same VM.
- **PM-local**: A task and its required data are on two co-located VMs.

At the beginning of running a MapReduce job, vLocality launches a topology configuration process that identifies the structure of the underlying virtualized clusters from a configuration file. Then a VM is associated with a unique ID in the format of *rack-PM-VM* such that the degree of locality can easily be obtained. A vScheduling algorithm (Algorithm 1) then schedules the tasks in a virtualized MapReduce cluster based on the newly designed priority levels as follows. When a VM has free slots, it requests new tasks from the master node through heartbeat, and the task scheduler on the master node assigns tasks according to the following priority order: PM-local if this VM has no DataNode (VM-local if this VM has a DataNode), rack-local, and off-switch.

The above high-level description provides a sketch of vLocality's workflow. Implementing it in real-world MapReduce systems, however, is nontrivial. In particular, we need to modify the original priority level as well as the corresponding scheduling policy, calling for careful examination of the whole Hadoop package to identify the related classes and methods. We also need to ensure that our modifications are compatible with other modules.

To demonstrate the practicability of vLocality and understand its practical performance, we have implemented a prototype of vLocality based on Hadoop 1.2.1. First, we modify the main configuration file `core-site.xml`, and add two more configuration files to declare the topology information of the underlying virtualized cluster. Second, we revise the task scheduling algorithm based on the proposed four-level locality design. In particular, we replace the original `NODE-LOCAL` by `PM-LOCAL` and `VM-LOCAL` depending on whether a task is launched on the local VM or on a co-located VM. The task priority is then updated by the method `getPhysicalMachine()` to PM-local (0), VM-local (1), rack-local (2), and off-switch (3), based on the distance between data (DataNode) and computation (TaskTracker). Second, we modify the task scheduling algorithm to be virtualization aware, which is mainly implemented in `JobInProgress.java` and `JobQueueTaskScheduler.java`. The task scheduler in vLocality first assigns the VM-local and PM-local tasks through `obtainVmOrPmLocalMapTask()`, then the rack-local tasks through `obtainRackLocalMapTask()`, and finally other tasks through `obtainNonLocalMapTask()`, as shown in Fig. 3a. All these three methods are encapsulated in a general method, `obtainNewMapTask()` in the file `JobInProgress.class`.

1. when the task scheduler receives a heartbeat from node v:
2. **if** *v* has a free slot **then**
3. sort the pending tasks according to the priority policy and obtain a list *T*
4. schedule the first task to node *v*
5. **end if**

ALGORITHM 1. vScheduling: Task scheduling in vLocality MapReduce systems.

Performance Evaluation

In this section, we evaluate the performance of vLocality based on a virtualized cloud testbed. Our testbed consists of eight PMs that are interconnected through a gigabit switch. We use one dedicated PM as the master node, and create three DomainU VMs on each of the remaining seven VMs. The setup of each VM is the same as that described earlier. Hence, our virtualized MapReduce cluster has 22 nodes in total: one physical master node and 21 virtual slave nodes. We compare vLocality with two other systems, *Default* and *HDFS-only*. Default runs the standard Hadoop 1.2.1 system with one DataNode on each virtual slave node, which serves as the baseline; HDFS-only adopts the one-DataNode-per-PM setting with the original task scheduling algorithm. We select three widely used Hadoop benchmark applications, `Sort`, `ShortestPath`, and `PageRank`. We use the 3.5 GB Wikipedia data as the input for `Sort`, and process the data to the input format (directed graph) of `ShortestPath` and `PageRank`.[7] The `PageRank` application ranks the vertices according to the PageRank algorithm, and the `ShortestPath` application finds the shortest path for all pairs of vertices in the input file.

It is worth noting that in all the experiments, all the file blocks are evenly distributed on all VMs, and thus on all PMs as well. In each experiment, the file access pattern is uniform, which means that each file block is accessed once. In this scenario, the default system already achieves the optimal node-local data locality, as discussed in [4, 9]. Since vLocality takes on the major efforts in minimizing the interference of co-located VMs, as well as differentiating between PM-local and rack-local, our testbed platform, where all the nodes are connected with one switch, focuses

[7] Each URL in the file is represented by a vertex, and each hyperlink is represented by a directed edge.

on improvements at the rack level. The results are representative, and the conclusion can be extended to larger systems since cross-rack scheduled tasks only account for a marginal portion of the total tasks [15].

Job Finish Time

We first compare the job finish time of different systems. For each system, we run each benchmark application five times, and plot the average job finish time in Fig. 4.

It is observed that vLocality significantly improves the performance of all the selected MapReduce applications over the other systems. Compared to Default, HDFS-only improves the job finish time by 9.6 percent on average for `Sort`, which again verifies the effectiveness of revising the DataNote placement strategy. vLocality, by adapting the task scheduling algorithm to be virtual aware, can further improve the job finish time by 16.3 percent on average compared to HDFS-only (24.3 percent when directly compared to Default).

The improvements on `ShortestPath` and `PageRank` are less significant (16.2 percent 12.3 percent compared to Default, respectively), since the number of reducers in these two applications is less than that in `Sort`, and thus the reduce phase, in which data locality has less impact, accounts for most of the running time. Further, these two applications involve iterative steps, which incur a lot of communication overhead.

Since data locality is mainly related to the map phase, we plot of the empirical CDF of the map task finish time in Fig. 5, which gives a much clearer picture of the impressive improvements of vLocality over Default. We can see that the system design has a remarkable impact on the finish time of individual map tasks. Taking `Sort` as an example, in the Default system, all the map tasks take more than 20 s, and nearly 30 percent of the tasks need more than 40 s. On the other hand, in both HDFS-only and vLocality, more than 70 percent of the tasks can be finished in less than 18 s, indicating that the reduced number of DataNodes can effectively mitigate the interference and speed up task execution. The remaining tasks, accounting for about 20 percent of the total, have divergent finish time distributions in HDFS-only and vLocality: all the remaining tasks can be finished within 40 s in vLocality, while most of them need more than 40 s in HDFS-only, implying that vScheduling significantly reduces rack-local (cross-PM) tasks.

Data Locality

We further calculate the distribution of different degrees of data locality for each system running the benchmark applications, and report the results of `Sort` in Table 1. The results are similar for `ShortestPath` and `PageRank`, which are omitted for the sake of conciseness.

Table 1 explains the advantages of vLocality over Default and HDFS-only. In Default, most tasks are VM-local, a few are rack-local, and PM-local cases are very rare. Concurrent running VM-local tasks on co-located VMs will incur serve inter-VM interference, leading to increased task finish time. HDFS-only successfully reduces VM-local tasks by reducing the number of DataNodes per PM, but it incurs many rack-local tasks since

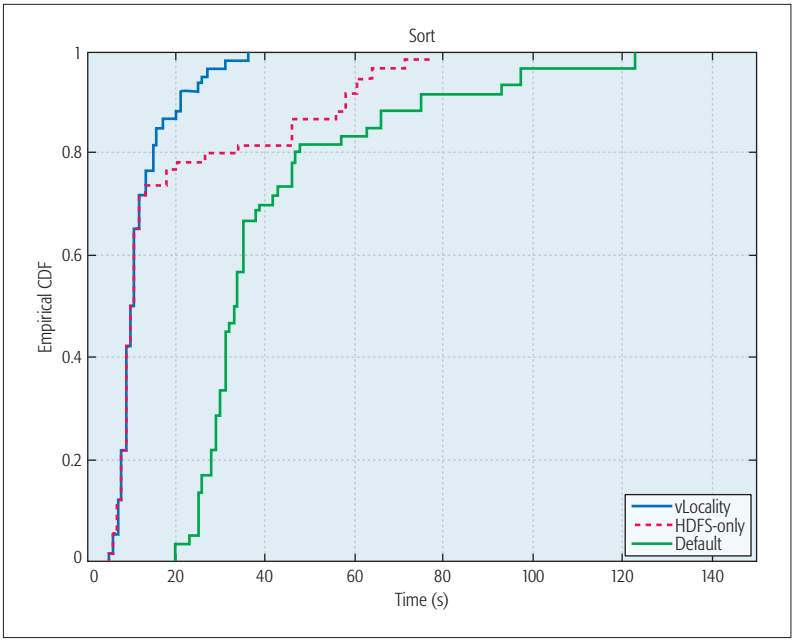

FIGURE 5. Empirical CDF of map task finish time of different systems.

the original task scheduler regards PM-local and rack-local as identical. In vLocality, most tasks are assigned to the appropriate VMs, minimizing the non-necessary rack-local tasks. It is worth noting that vLocality cannot completely eliminate the rack-local tasks. To identify the root cause, we analyze the log files and find that in some cases, the VMs on one PM are all busy and have no free slots, while some VMs on other PMs have free slots. Hence, the task scheduler has to allocate the pending tasks, which are rack-local to these VMs; otherwise, the slot resources would be wasted.

Further Discussion

Data locality is critical to the performance of MapReduce tasks, and there have been significant studies toward improving data locality [3]. Scarlett [4] and DARE [5] adopt a popularity-based strategy, which smartly place more replicas for popular file blocks. There have also been substantial efforts on directly scheduling tasks close to data [10, 12, 13].

These works on data locality have yet to address the important challenges of VMs in a public cloud. In the virtualized environment, a location-aware data allocation strategy was proposed in [9], which allocates file blocks across all PMs evenly, with replicas located in different PMs. DRR [14] enhances locality-aware task scheduling through dynamically increasing or decreasing the computation capability of each node. An interference and locality-aware (ILA) scheduling strategy has been developed for virtualized MapReduce clusters, using a task performance prediction model to mitigate inter-VM interference and preserve task data locality [15].

Our vLocality improves the data locality in a virtualized cloud environment by observing that directly applying conventional data locality strategies in PM clusters to VM clusters (i.e., achieving complete VM-locality) can be harmful to MapReduce performance given the shared

We plan to conduct a more detailed profiling analysis to identify the potential performance bottleneck of the vLocality design, and investigate both offline solutions that allocate different amounts of resources based on workloads, as well as online solutions that dynamically adjust the capabilities of VMs through the Xen control interfaces.

	VM-Local	PM-local	rack-local
Default	90.0%	1.7%	8.3%
HDFS-only	31.7%	38.3%	30.0%
vLocality	25.0%	71.7%	3.3%

TABLE 1. Degrees of data locality in different systems.

resource contention. We have revised the underlying data placement strategy and accordingly propose a customized virtual-aware task scheduling algorithm. The existing improvements (e.g., Delay Scheduling [10], DRR [14], and interference-aware scheduling [15]) can be further incorporated into vLocality after being customized for the new architecture of vLocality. We plan to work on this issue in future work.

The virtualization performance overhead in cloud scenarios has attracted a lot of attention from researchers [8]. In general, the causes of overhead stem from different levels of cloud infrastructure: from single-server virtualization, a single-site mega data center, to multiple geo-distributed data centers. In this article, we investigate the single-server virtualization overhead for MapReduce-based big data processing, and vLocality is also beneficial to mitigate the overhead on the data center scale by reducing cross-sever traffic.

The dynamics of virtualized cloud platforms (e.g., VM migration [8, 16]) is also very important for load balancing, power saving, and fault tolerance. In big data processing, VM migration can be very costly since large volumes of data also need to be migrated besides VM images. The data center network would be overloaded during migration, and the performance interference on migration source and destination PMs can be severe. vLocality is able to reduce the migration overhead since migrating a VM without DataNode incurs much less data transfer than migrating a fully functional VM, while the data availability is not affected. The iAware strategy [16] can be also adapted to vLocality (e.g., considering the data locality issue when selecting destination PMs) to further minimize the overhead of live migration.

Conclusion

As an important practice to improve MapReduce's performance, data locality has been extensively investigated in physical machine clusters. In this article, we show strong evidence that the conventional efforts on improving data locality can have a negative impact on typical MapReduce applications in virtualized clouds. We suggest adapting the existing storage architecture to be virtual-machine-aware, which offloads a large portion of congested disk I/O to the highly efficient network I/O with memory sharing. This new storage design demands revision of the traditional three-level data locality, as does the task scheduling. We have developed vLocality, a systematic solution to improve data locality in virtualized MapReduce clusters, and demonstrate the superiority of vLocality against state-of-the-art systems.

We plan to further optimize the two basic components of vLocality: for storage design, we can incorporate the file popularity issue to balance the workloads of individual DataNodes; for task scheduling, we may incorporate other advanced strategies, such as Delay Scheduling [10], into the virtualized environment. We will also investigate the possibility of more flexible resource allocation in vLocality. Given that not all VMs have DataNodes, the workloads of different types of VMs can be different, leading to a heterogeneous environment. We plan to conduct a more detailed profiling analysis to identify the potential performance bottleneck of the vLocality design, and investigate both offline solutions that allocate different amounts of resources (e.g., CPU and memory) based on workloads, as well as online solutions that dynamically adjust the capabilities of VMs through the Xen control interfaces.

References

[1] X. Yi *et al.*, "Building A Network Highway for Big Data: Architecture and Challenges," *IEEE Network*, vol. 28, no. 4, July/Aug. 2014, pp. 5–13.
[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, Jan. 2008, pp. 107–13.
[3] Z. Guo, G. Fox, and M. Zhou, "Investigation of Data Locality in MapReduce," *Proc. IEEE/ACM CCGRID*, 2012.
[4] G. Ananthanarayanan *et al.*, "Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters," *Proc. EuroSys*, 2011.
[5] C. L. Abad, Y. Lu, and R. H. Campbell, "Dare: Adaptive Data Replication for Efficient Cluster Scheduling," *Proc. IEEE CLUSTER*, 2011.
[6] S. Ibrahim *et al.*, "Evaluating MapReduce on Virtual Machines: The Hadoop Case," *Proc. CloudCom*, 2009.
[7] Y. Yuan *et al.*, "On Interference-Aware Provisioning for Cloud-Based Big Data Processing," *Proc. IEEE/ACM IWQoS*, 2013.
[8] F Xu *et al.*, "Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions," *Proc. IEEE*, vol. 102, no. 1, Jan. 2014, pp. 11–31.
[9] Y. Geng, S. Chen *et al.*, "Location-Aware MapReduce in Virtual Cloud," *Proc. ICPP*, 2011.
[10] M. Zaharia *et al.*, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," *Proc. EuroSys*, 2010.
[11] P. Barham *et al.*, "Xen and the Art of Virtualization," *Proc. ACM SOSP*, 2003.
[12] X. Zhang *et al.*, "Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments," *Proc. IEEE ISPA*, 2011.
[13] M. Hammoud and M.F. Sakr, "Locality-Aware Reduce Task Scheduling for MapReduce," *Proc. IEEE CloudCom*, 2011.
[14] J. Park *et al.*, "Locality-Aware Dynamic VM Reconfiguration on MapReduce Clouds," *Proc. HPDC*, 2012.
[15] X. Bu, J. Rao, and C. Xu, "Interference and Locality-aware Task Scheduling for MapReduce Applications in Virtual Clusters," *Proc. HPDC*, 2013.
[16] F. Xu *et al.*, "iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud," *IEEE Trans. Computers*, vol. 63, no. 12, Dec. 2014, pp. 3012–25.

Biographies

Xiaoqiang Ma (maxiaoqiang@hust.edu.cn) received his B.Eng. degree from Huazhong University of Science and Technology, China, in 2010; he received his M.Sc. and Ph.D. degrees from Simon Fraser University, Canada, in 2012 and 2015, respectively. He is currently an assistant professor in the School of Electronic Information and Communication at Huazhong University of Science and Technology. His current research interests are in the areas of wireless networking, multimedia, cloud, and big data.

Xiaoyi Fan (xiaoyif@sfu.ca) received his B.E.ng degree from Beijing University of Posts and Telecommunications, China, in 2013, and his M.Sc. degree from Simon Fraser University in 2015. He is now a Ph.D. student in the School of Computing Science, Simon Fraser University. His areas of interest are cloud computing, big data, and wireless networks.

Jiangchuan Liu (jcliu@sfu.ca) is a University Professor in the School of Computing Science, Simon Fraser University, and an NSERC E.W.R. Steacie Memorial Fellow. He is an EMC-Endowed Visiting Chair Professor of Tsinghua University, Beijing, China (2013–2016). From 2003 to 2004, he was an assistant professor at the Chinese University of Hong Kong. He received his B.Eng. degree (cum laude) from Tsinghua University in 1999, and his Ph.D. degree from the Hong Kong University of Science and Technology in 2003, both in computer science. He is a co-recipient of the inaugural Test of Time Paper Award of IEEE INFOCOM (2015), the ACM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), the *ACM Multimedia* Best Paper Award (2012), the IEEE GLOBECOM Best Paper Award (2011), and the IEEE Communications Society Best Paper Award on Multimedia Communications (2009). His students received the Best Student Paper Award of IEEE/ACM IWQoS twice (2008 and 2012). His research interests include multimedia systems and networks, cloud computing, social networking, online gaming, big data computing, wireless sensor networks, and peer-to-peer networks. He has served on the Editorial Boards of *IEEE Transactions on Big Data*, *IEEE Transactions on Multimedia*, *IEEE Communications Surveys & Tutorials*, *IEEE Access*, the *IEEE Internet of Things Journal*, *Computer Communications*, and *Wiley Wireless Communications and Mobile Computing*. He is the Steering Committee Chair of IEEE/ACM IWQoS from 2015 to 2017.

Hongbo Jiang [SM] (hongbojiang2004@gmail.com) received his B.S. and M.S. degrees from Huazhong University of Science and Technology. He received his Ph.D. from Case Western Reserve University in 2008. After that he joined the faculty of Huazhong University of Science and Technology, where he is a full professor and Dean of the Department of Communication Engineering. His research concerns computer networking, especially algorithms and protocols for wireless and mobile networks. He is serving as an Associate Editor of *IEEE Transactions on Mobile Computing*, *ACM/Springer Wireless Networks*, and *Wiley Security and Communication Networks*, and as an Associate Technical Editor of *IEEE Communications Magazine*.

Kai Peng (eikaipeng2015@gmail.com) received his B.S., M.S., and Ph.D. degrees from Huazhong University of Science and Technology in 1999, 2002, and 2006, respectively. He is now with the faculty of the same university as an associate professor. His current research interests are in the areas of wireless sensor networking and big data processing.