Appears in the 21st International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'12)

Locality-Aware Dynamic VM Reconfiguration on MapReduce Clouds

Jongse Park, Daewoo Lee, Bokyeong Kim, Jaehyuk Huh, Seungryoul Maeng

Computer Science Department, KAIST
Daejeon, Korea

{jspark, dwlee, bokyeong, jhuh, and maeng}@calab.kaist.ac.kr

ABSTRACT

Cloud computing based on system virtualization, has been expanding its services to distributed data-intensive platforms such as MapReduce and Hadoop. Such a distributed platform on clouds runs in a virtual cluster consisting of a number of virtual machines. In the virtual cluster, demands on computing resources for each node may fluctuate, due to data locality and task behavior. However, current cloud services use a static cluster configuration, fixing or manually adjusting the computing capability of each virtual machine (VM). The fixed homogeneous VM configuration may not adapt to changing resource demands in individual nodes.

In this paper, we propose a dynamic VM reconfiguration technique for data-intensive computing on clouds, called *Dynamic Resource Reconfiguration (DRR)*. DRR can adjust the computing capability of individual VMs to maximize the utilization of resources. Among several factors causing resource imbalance in the Hadoop platforms, this paper focuses on data locality. Although assigning tasks on the nodes containing their input data can improve the overall performance of a job significantly, the fixed computing capability of each node may not allow such locality-aware scheduling. DRR dynamically increases or decreases the computing capability of each node to enhance locality-aware task scheduling. We evaluate the potential performance improvement of DRR on a 100-node cluster, and its detailed behavior on a small scale cluster with constrained network bandwidth. On the 100-node cluster, DRR can improve the throughput of Hadoop jobs by 15% on average, and 41% on the private cluster with the constrained network connection.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: [Organization and Design] Distributed System; C.2.4 [**Computer-communication Networks**]: Distributed Systems

HPDC'12, June 18–22, 2012, Delft, The Netherlands.

General Terms

Management, Design

Keywords

Cloud computing, virtual clusters, MapReduce

1. INTRODUCTION

Recently, cloud computing has been replacing traditional privately owned clusters, as it can provide high efficiency from the economies of scale and elastic resource provisioning. Such cloud computing has been expanding its services to data-intensive computing on distributed platforms such as MapReduce [10], Dryad [12], and Hadoop [2]. In such distributed platforms on clouds, physical machines are virtualized, and a large number of virtual machines (VMs) form a virtual cluster. A data-intensive platform runs on the virtual cluster instead of a traditional physical cluster. Such a virtual cluster can provide a highly flexible environment, which can scale up and down accommodating changing computation demands from various users. Cloud providers consolidate virtual clusters from different users into a physical data center, to maximize the utilization of resources.

Current virtual clusters for data-intensive computing can support the flexibility of selecting the type of computing nodes and the number of nodes in a cluster, when the cluster is configured. Users can choose the most appropriate virtual cluster configuration to meet their computational requirements. Although such a static configuration of each virtual machine in the cluster can still provide better flexibility than clusters with physical machines, the static configuration cannot satisfy dynamically changing computing demands during the life time of a virtual cluster. To adapt to changing demands on each VM in a virtual cluster, each VM may be dynamically reconfigured. Current virtualization techniques can support such a dynamic reconfiguration of each virtual machine using resource *hot-plugging*. As long as physical resources are available, each virtual machine can be assigned with more virtual CPUs and memory while the virtual machine is running. However, the currently available cloud services, such a dynamic reconfiguration of VM is not available,

In the distributed data-intensive computing platforms, resources required for each node may not be uniform. In such platforms, a user job is decomposed into many small tasks, and the tasks are distributed across computing nodes in the cluster. One of the most critical reasons for uneven resource usages is data locality. Input data are dis-

tributed across computing nodes using distributed file systems, such as Google File System (GFS)[9] or Hadoop File Systems (HDFS). Depending on whether a task is assigned to a node with its data (*local task or non-local task*), the execution time of the task may differ significantly. Prior studies showed that data locality affects the throughput of Hadoop jobs significantly, and they improve the locality by assigning tasks to the nodes with the corresponding data as much as possible [19]. However, to improve locality, when a task is scheduled, the computing node with the corresponding data must have available computing slots to process the task. If a computing slot is not available, the task must be scheduled to a remote node, which must transfer necessary data from another node for the task.

In this paper, we propose a dynamic VM reconfiguration technique, called *Dynamic Resource Reconfiguration (DRR)* for virtual clusters running the Hadoop platform. The technique can change the configuration of virtual computing nodes dynamically to maximize the data locality of tasks. The proposed technique increases the computing resource of a VM, if the next task has its data on the VM. Simultaneously, an idle virtual CPU is removed from another VM in the same virtual cluster, so that the size of the virtual cluster remains constant to provide a constant cost for the user. This dynamic reconfiguration of each VM improves the overall job throughput by improving data locality, while the total virtual CPUs in the cluster remain unchanged.

However, to add a virtual CPU to a VM for a newly scheduled local task, the physical system running the VM must have additional available CPU resources. Cloud providers may reserve some headroom for such a temporary increase of CPU demand in each physical system. Our first DRR scheduler, called *Synchronous DRR*, assumes the availability of such extra CPU resources in each physical system. Alternatively, we also propose a DRR scheduler, called *Queue-based DRR*, which can eliminate the CPU headroom completely. The scheduler coordinates the allocation and deallocation of virtual CPUs from different physical systems. Virtual clusters sharing the physical cluster, can exchange CPU resources, by deallocating virtual CPUs from a node without local tasks, and by allocating virtual CPUs to a node with pending local tasks. Our results show that such an exchange incurs only minor delays in executing tasks.

To the best of our knowledge, this paper is one of the first studies to dynamically reconfigure individual VMs in a virtual cluster running distributed data-intensive platforms. Each user may just limit the total resource size of a virtual cluster, while the configuration of each VM is dynamically determined by changing resource demands for the VM. Using a 100-node Amazon EC2 [1] cluster, we evaluate the potential performance improvement by locality-aware dynamic VM reconfiguration. With the Hive benchmark [5] running on the Hadoop and HDFS platforms, locality-aware reconfiguration can improve the overall throughput by 15% on average. On a small scale private cluster with a limited network bandwidth, dynamic reconfiguration can improve the throughput by 41% on average.

Another source for uneven data demands is that different jobs may have different resource usages. For example, some jobs require more CPU resources, while others need more memory or I/O bandwidth. Although this paper focuses only on mitigating load imbalance caused by data locality, DRR can be generalized to other dynamic resource imbalance among VMs, adjusting the configuration of each VM for changing demands.

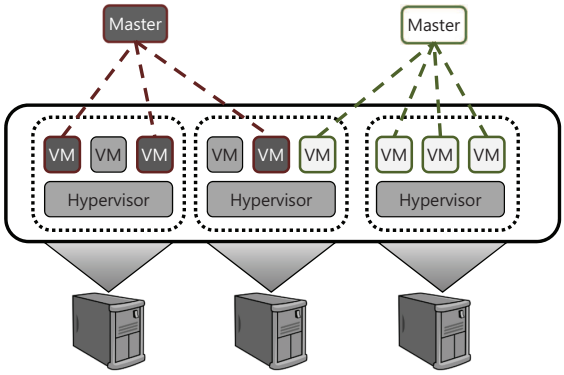

Figure 1: MapReduce on virtualized environments

The rest of this paper is organized as follows. Section 2 presents motivation for dynamic VM reconfiguration to improve data locality in data-intensive platforms. Section 3 describes the overall architecture of DRR and its reconfiguration policies. Section 4 evaluates the effectiveness of DRR with two different cluster environments. Section 5 presents prior work on VM reconfiguration and data-intensive platforms, and Section 6 concludes the paper.

2. MOTIVATION

2.1 Dynamic VM Reconfiguration

In cloud computing, system virtualization allows flexible resource management by allocating virtual machines, instead of physical systems, to cloud users. With virtualization, each user has an isolated secure computing environment, even though multiple users may share physical resources to improve the utilization of physical systems. System virtualization has been extended to a virtual cluster, which consists of multiple virtual machines, for distributed computing platforms. Instead of using physical systems directly, data and computation in such data-intensive platforms are distributed across a large number of virtual machines. Figure 1 depicts multiple virtual clusters running MapReduce platforms sharing the same physical cluster.

To configure virtual clusters, current public or private clouds use a homogeneous static configuration of virtual machines. When a virtual cluster is created, a user selects a type of virtual machines with a fixed number of virtual cores and a fixed memory size, and determines the number of virtual machines, by considering the overall cost of using the virtual cluster. One drawback of such a static configuration is that the resources required for each virtual machine may fluctuate during the life time of the virtual cluster. A virtual machine may require more CPU resources while another virtual machine needs more memory. Such a dynamic imbalance of resources in individual virtual machines, leads to the overall inefficiency of cluster resources.

In traditional data-intensive computing on physical clusters, it is not possible to change physical resources dynamically, as each physical system has a fixed amount of physical resources. In such physical clusters, numerous studies for data-intensive computing have tried to maximize the uti-

lization of fixed physical resources by efficiently distributing loads on the nodes [10, 12, 2, 16, 4, 3]. A job is partitioned into small tasks, and evenly distributed to across computing nodes for load balancing. However, such a perfect load balancing may not be possible, as different jobs or tasks have different resource requirements. Furthermore, as will be discussed in the next section, data locality often makes a naive uniform distribution of tasks, inefficient for the overall throughput of the platform.

However, virtualization opens a novel opportunity to reconfigure virtual machines constituting a virtual cluster. Multiple virtual clusters share a set of physical machines, and within a physical system, multiple virtual machines for the same or different users co-exist. Exploiting the flexibility of virtualization, resources allocated for each virtual machine can be dynamically reconfigured by *resource hot-plugging*. For example, the number of virtual CPUs for a virtual machine can change while the virtual machine is running, or the size of allocated memory can also change with ballooning techniques[18].

The dynamic reconfiguration of virtual machines allows resources to be provisioned to demanding nodes. This can make traditional load balancing less critical in distributed platforms, as resources in each VM become flexible. This leads to a shift of cloud service from provisioning of a fixed set of virtual machines, to a cluster-level resource provisioning. Users just need to select the total amount of resources for a given cluster, and each virtual machine can be reconfigured during the runtime to maximize the performance of users' workloads.

There are two important factors causing unbalanced loads in distributed data-intensive platforms. Firstly, data locality requires flexible CPU resources in computing nodes. For load balancing, incoming task must be evenly distributed across different nodes. However, such a pure load-balanced scheduling of tasks can lead to the ignorance of data locality, causing expensive data transfer for non-local tasks. To maintain data locality, the virtual machine, which has data for an incoming task, may increase its computational capability temporarily to process the task.

Secondly, each MapReduce job or task has a different resource requirement. For example, certain tasks require more CPUs and others require more memory or I/O bandwidth. Users cannot predict the resource requirement precisely when a virtual cluster is configured. If virtual machines can be dynamically reconfigured for tasks, the cloud provider can relieve users from the burden of selecting the most efficient cluster configuration. In this paper, we focus on the data locality problem for VM reconfiguration, leaving the generalized VM reconfiguration as future work. In the next section, we elaborate the data locality problem in distributed data-intensive platforms.

2.2 Data Locality in MapReduce

The data locality problem occurs in MapReduce platforms since input data are distributed in computing nodes, using a distributed file system, such as Google File System [9] or Hadoop File System. A user job is partitioned to tasks which process a block of data. The block size is commonly configured to 64MB or 128MB. The entire data set is also partitioned at the block granularity and is distributed in the computing nodes. To improve the reliability and locality of

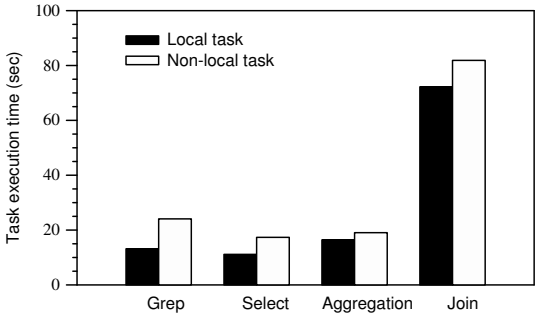

Figure 2: Task execution times for workloads at a 100-node cluster

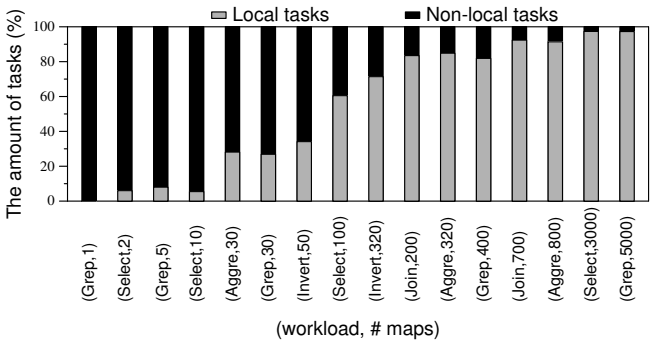

Figure 3: Data locality for various workloads with different numbers of map tasks at a 100-node cluster

input data, they are often replicated and stored in multiple computing nodes.

A MapReduce job is composed of three phases, map, shuffle, and reduce. The master schedules map and reduce tasks to free computing slots identified by heartbeats from slave nodes. Each slave sends a heartbeat every few seconds which announces that it can handle a new task if it has a free slot. Each slave node has a fixed number of map and reduce slots, which are proportional to the available computation capability, commonly one or two slots per core.

After receiving a heartbeat, the master assigns a new task to the slave node according to its scheduling policy. To improve data locality, the master scheduler attempts to assign a map task, which processes the data stored in the node with an available slot. However, it is not always possible to find such a *local* task to the node. As reduce tasks commonly receive data equally from all nodes, the data locality is less critical for reduce tasks than for map tasks.

If data locality is not satisfied in the node with a free slot, the node should read input data for computation through network, incurring significant network overheads. Such a remote data transfer for a map task, not only delay the execution time of the map task, but also increase network traffic in the cluster, negatively affecting other tasks running in the cluster. Although reduce tasks do not exhibit a significant data locality, as it receives intermediate data from all the nodes, the reduced network traffic from localized map tasks, can indirectly improve the performance for reduce tasks too. Therefore, it is critical to assign tasks to the nodes which have their input data, satisfying data locality.

Figure 2 shows the difference of execution times between *local* and *non-local* tasks. Local tasks get their input data

from local disks during the map phase, while non-local tasks require data transfers from remote nodes. The results are measured from a 100-node cluster from Amazon EC2. The details of the methodology are described in Section 4.1. Depending on the locality of input data, task execution times can increase by as much as 53% in the `grep` workloads. The `select` workloads also exhibit a significant increase of execution times, if a map task must receive its input data from a remote node.

Figure 3 presents the ratios of local tasks for various sizes of MapReduce jobs. As the size of jobs, or the number of map tasks, increases, the chance to schedule map tasks to local VMs increases, since the scheduler has many candidate local map tasks when a slot becomes free in a VM. However, if a job consists of a small number of map tasks, scheduling the tasks to local VMs becomes very difficult. In commercial usage patterns as shown by Zaharia et al [19], such small jobs account for the majority of jobs running in Facebook. Their study showed that as a significant portion of real world workloads are small jobs, with low data locality, improving the data locality for such jobs is critical for the overall cluster performance. As the number of nodes increases, the locality problem for small jobs will become worse, as the data set is distributed in more nodes.

Fundamentally, the reason why the data locality issue becomes a critical problem on MapReduce is that a node of a MapReduce cluster has a dual role of computing node as well as data node. Load balancing for computation often conflicts with data locality. Dynamic VM reconfiguration mitigates such conflicts between computation load balancing and data locality, as the reconfiguration can transfer computational resources to the node with a pending task and the necessary data for the task. Virtualization enables such transfers of computation resources from a VM to another VM by dynamically adjusting virtual CPU resources in two VMs.

3. ARCHITECTURE

In this section, we describe a dynamic reconfiguration mechanism for virtual clusters to support locality-aware resource scheduling. We propose two schemes for the dynamic VM reconfiguration. The first scheme, synchronous DRR, can reconfigure VMs only when there are free CPU resources in physical systems. To support the first scheme, the cloud provider may need to reserve unassigned CPU slots in each physical system. The second scheme, queue-based DRR, does not require such headroom in CPU resources. We first describe the necessary modifications to virtual clusters to provide dynamic reconfiguration, and present the two schemes.

3.1 Overview

To support dynamic VM reconfiguration for Hadoop platforms, the cloud provider must offer cluster-level pricing options to allow users to choose the base VM configuration and the number of VMs. However, the configuration of a VM can be dynamically adjusted with more or less virtual CPUs than the base VM configuration, while the total cores assigned to the cluster does not change. The dynamic resource reconfiguration (DRR) architecture adds or removes virtual cores in VMs to maximize the data locality of the Hadoop platform.

To implement DRR, two new components must be added to the current virtual cluster environments. The first component is *Reconfiguration Coordinator (RC)* which is a reconfiguration manager sending allocation and deallocation requests to the hypervisor running on each physical machine. A single RC is created for each virtual cluster and the master node of each virtual cluster runs the RC. If RC decides that a virtual machine in the cluster needs resource reconfiguration, RC sends an allocation or de-allocation request to *Machine Resource Manager (MRM)* running in the physical system, where the VM to be reconfigured is running on. After receiving requests from RC, MRM re-assigns the requested virtual CPUs to the VM. MRM makes requests to the hypervisor of the physical machine, to hot-plug or to un-plug virtual CPUs for VMs running on the system.

Algorithm 1 Base Fair Hadoop Scheduler [19]

```
1: when a heartbeat is received from node n:
2: if n has a free slot then
3:     sort jobs in increasing order of the number of running tasks
4:     for j in jobs do
5:         if j has unlaunched task t with data on n then
6:             launch t on n
7:         else if j has unlaunched task t then
8:             launch t on n
9:         end if
10:    end for
11: end if
```

In this section, we propose two possible implementations of DRR, *Synchronous DRR* and *Queue-based DRR*. In the synchronous DRR, a task is assigned to a VM only when the physical system running the VM has an available core to allocate for the VM. Such synchronous reconfiguration requires free CPU resources in each system on the cloud. However, in the queue-based DRR, a task can be queued to a VM, even if the VM does not have an available virtual core immediately. The queue-based DRR increases the possibility of maintaining locality by waiting for a possibly short period time until the locality-matched VM has an available core. To support the queue-based DRR, MRM maintains two queues, allocation and deallocation queues (AQ and DQ). When the scheduler assigns a task to a VM, but the VM does not have an available slot, the task is appended to the allocation queue in MRM, waiting for an available core. If the VMs running the physical system have an idle core, MRM of the system appends the idle core to the deallocation queue.

Two proposed DRR implementations are both based on a naïve Hadoop scheduler for fairness [19]. The scheduler provides fairness among user jobs, while locality is also marginally supported. When the scheduler receives a heartbeat from a node (or VM in a virtual cluster) and the node has a free slot, it picks a job which has the lowest number of running tasks in the cluster, to guarantee the fairness among jobs. For locality, among the tasks in the picked job, the scheduler attempts to select a task, whose input data is in the newly freed node (local task for the node). If the scheduler cannot find a local task for the node, it attempts to find a task whose data is in the node of the same rack. Neither tasks are found for the picked job, the scheduler picks any task for the free node. Although the Hadoop scheduler distinguishes local and rack-local tasks, common cloud computing environments often do not support the distinction between local and rack-local tasks, as the cluster topology

Algorithm 2 Synchronous DRR Algorithm

```
1: when a heartbeat is received from node n:
2: if n has a free slot then
3:     sort jobs in increasing order of the number of running tasks
4:     for j in jobs do
5:         if j has unlaunched task t with data on n then
6:             launch t on n
7:         else if j has unlaunched task t then
8:             find node set s storing data of t
9:             pick a node m from s
10:            send resource allocation request to MRM{P_m}
11:            send resource de-allocation request to MRM{P_n}
12:            launch t on m
13:        end if
14:    end for
15: end if
```

is not revealed by the provider. Due to the limitation, this paper distinguishes a task only into either a local or remote task. Algorithm 1 describes the base fair scheduler, which guarantees jobs fairness with limited support for locality.

3.2 Synchronous DRR

Synchronous DRR can dynamically add a virtual core to a VM, if the VM has input data for a pending task. One restriction of the synchronous DRR is that it can add a core to a VM, only when there is a free CPU resource in the physical system (target system) the VM is running on. If the target does not have any available CPU resource, adding a virtual core to the VM will reduce the CPU shares assigned to the other virtual cores of the same VM or other VMs. Assigning more virtual cores than the available CPU resources can negatively affect other VMs and violate Service Level Agreement (SLA) for other users. Therefore, it should not be allowed to add a virtual core to a VM, if the physical system does not have an extra core to accommodate the virtual core to be added.

The synchronous DRR scheduler is based on the base fair scheduler to support both dynamic reconfiguration and fairness among user jobs. Algorithm 2 presents the synchronous DRR algorithm, extending the base fair scheduler discussed in the previous section. In the synchronous DRR algorithm, when a VM (*source VM*) has a free slot, the scheduler first picks a job for fairness, and attempts to pick a local task for the VM. If no local task is found for the VM, it picks any task and finds the VM (*target VM*) with the input data of the selected task. Once the target VM is selected, the scheduler in RC sends a CPU allocation request to the target system. At the same time, the scheduler also sends a CPU deallocation request to the source system, since the source VM, which has reported a free slot, does not have a task to schedule. The CPU allocation and deallocation requests are sent to MRMs in the source and target systems, and MRMs request the hypervisors to add or remove virtual cores. If no free core is available in the target system, the picked task is scheduled to the original source VM, violating the data locality of the task. By the coordination of RC and two MRMs in the source and target VMs, a core resource is transferred from the source VM to the target VM.

The limitation of the synchronous DRR algorithm is that it is effective only when the system running the target VM has an available core slot. The cloud provider can increase the chance of such a readily available CPU resource for temporary locality guarantee in various ways. The cloud provider may leave a small portion of core resources as a headroom for dynamic CPU resource demands. As the number of cores in a system is expected to increase to tens of cores in future systems, reserving one or two cores for the headroom may not degrade the overall throughput. In the next section, we relax such a constraint of the synchronous DRR using per-system queues.

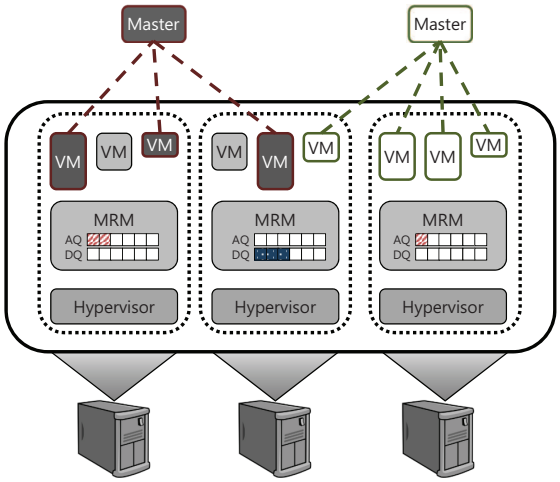

Figure 4: MapReduce with queue-based DRR

Algorithm 3 Queue-based DRR Algorithm

```
1: when a heartbeat is received from node n:
2: if n has a free slot then
3:     sort jobs in increasing order of number of running tasks
4:     for j in jobs do
5:         if j has unlaunched task t with data on n then
6:             launch t on n
7:         else if j has unlaunched task t then
8:             find node set s_d storing data of t which has entries
   on deallocation queue
9:             sort s_d in decreasing order of number of entries in
   deallocation queue
10:            if s_d is not ∅ then
11:                pick m at the top of sorted set s_d
12:            else
13:                find node set s_a storing data of t
14:                sort s_a in increasing order of number of entries
   in allocation queue
15:                pick m at the top of sorted set s_a
16:            end if
17:            send resource allocation request to MRM{P_m}
18:            send resource de-allocation request to MRM{P_n}
19:            launch t on m
20:        end if
21:    end for
22: end if
```

3.3 Queue-based DRR

Resource reconfiguration is composed of resource allocation to the target VM and de-allocation from the source VM. In the synchronous DRR, an allocation and deallocation of a core must occur in a coupled manner. An advantage of the synchronous DRR is that a task is never delayed, as it is always assigned to a local VM, if possible, but to a remote VM otherwise. If these two virtual machines locate on the same physical machine, resource reconfiguration between these virtual machines could be done immediately, but

otherwise, allocating new resources in the physical machine of target VM may not be possible, if there is no available CPU resource.

However, to improve the chance to support locality, a task can be delayed until a core becomes available in the target node. Instead of synchronously deallocating and allocating cores in the source and target VMs, the queue-based DRR allows the decoupling of the two operations. If a VM has a free slot, it registers the free core to the deallocation queue (DQ) of the system. If a VM has a pending local task assigned by RC, the task is appended to the allocation queue (AQ) of the system. As soon as both the AQ and DQ of the same system (MRM) has at least an entry, VM reconfigurations occur in the system, deallocating a core from a VM, and allocating a core to another VM in the same system.

This delayed task scheduling occurs since the CPU resource cannot be transferred beyond the physical system boundary directly. Even if the system running the source VM has a free core, the computing resource cannot be directly available to the target VM. However, with multiple VMs sharing a physical system, the target system will soon have a free core, as a task finishes in one of the VMs, and a local task is not found for the VM.

In the queue-based DRR, such a queuing delay can be an important factor for cluster performance, as resource utilization might be degraded if handling request of the allocation and de-allocation queue is postponed due to a large queuing delay. In this paper, we use two schemes to reduce the queuing delay. For the schemes, RC must know the allocation and deallocation queue lengths in all MRMs. The first scheme attempts to schedule a task to the matching local VM in the system with the longest deallocation queue. Since the system has a pending deallocation request, the VM for the task can be reconfigured immediately. The second scheme, if there are no system with pending deallocation requests, schedules a task to the matching local VM with the shortest allocation queue. By selecting the system with the shortest allocation queue, the scheduler avoids increasing the allocation queues unnecessarily. In summary, task reconfiguration has to be done to the node with the largest number of deallocation entries, and the smallest number of allocation entries if there is no node with any deallocation entries.

Algorithm 3 describes the queue-based DRR algorithm. The algorithm checks the deallocation queue lengths of systems first (line 8 and 9), to assign a new task to the system with any available core. If not available, it checks the shortest allocation queue to spread tasks across systems in the cluster. This optimization reduces possible queue delays, and in Section 4.3, we evaluate the impact of such queue-aware schemes to improve the queue-based DRR.

The queue-based DRR mechanism overcomes the limitation of the synchronous DRR mechanism, eliminating the requirement for a CPU headroom in each physical system. Even with the queue-based DRR mechanism, the cloud provider can still add the CPU headroom in each machine, to further reduce possible short queue delays.

4. EVALUATION

In this section, we evaluate the locality and performance improvements by dynamic VM reconfiguration over the plain Hadoop platform. To evaluate the effectiveness of DRR in a large scale cluster, we use a 100-node Amazon EC2 virtual cluster, in addition to a small private cluster for detailed experiments. Firstly, we evaluate the potential performance improvements by DRR in the setups where additional cores are always available in each system. We use such a pseudo-ideal setup, since large-scale experiments on EC2 do not allow system modifications necessary to run a real queue-based DRR implementation. Secondly, we evaluate the queue-based DRR on our small scale private cluster.

Job Type	# jobs	# Maps	Intensivity
Grep	30	1	I/O
Select	20	2	I/O
Grep	15	5	I/O
Select	10	10	I/O
Aggregation	5	32	Communication
Grep	5	35	I/O
Inverted Index	5	50	Communication
Select	5	100	I/O
Inverted Index	4	150	Communication
Join	2	200	I/O
Aggregation	2	300	Communication
Grep	2	400	I/O
Join	1	600	Communication
Aggregation	2	800	Communication
Select	1	3000	I/O
Grep	2	5000	I/O

Table 1: Benchmarks on Amazon EC2

4.1 Evaluation Methodology

We evaluate our mechanisms in a cluster from Amazon Elastic Compute Cloud (EC2) as well as our private cluster with 6 physical machines. With Amazon EC2, we use 100 "High-CPU Extra Large" instances from the provider. It contains 8 virtual cores and 7GB memory, and supports high I/O performance. The topology information of the 100 nodes is not available. The 100-node EC2 cluster represents a relatively large scale virtual cluster, and we use the environment to show the potential performance and locality improvements by DRR. One drawback of the EC2 environment for our study is that it does not allow RC and MRM to run at the privileged level, as the controls over hypervisors are not available to guest users. Therefore, we configure our platform to use 6 cores per VM, and the rest two cores are reserved for dynamic reconfigurations to absorb temporary increases of core demands, as discussed in Section 3.2 for the synchronous DRR. However, the total number of virtual CPUs actually used for our workloads does not change, even if each node increases or decreases its core. The EC2 environment has the same effect as a synchronous DRR platform with two additional cores always available.

A physical machine included in the 6-node private cluster has an AMD Phenom 6-core processor, 16GB memory, and 1 disk. We allocate 5 virtual machines for each physical machine. Each virtual machine has 2 virtual cores, and 2GB memory. All of the physical machines are connected with a network switch, and we ran experiments on both 100Mbps and 1Gbps switches. The 100Mbps switch represents a constrained network bandwidth mimicking a constrained network bandwidth across multiple racks. A prior study use a similar 100Mbps switch to emulate the rack-to-rack network bandwidth [17]. With the small cluster size of 6 machines, the 1Gbps switch provides ample network bandwidth, and

Job Type	# jobs	# Maps	1Gbps	100Mbps
Grep	20	1	o	o
Select	15	2	o	o
Grep	10	5	o	o
Select	6	10	o	o
Grep	6	20	o	o
Aggregation	1	50	o	o
Select	2	100	o	o
Join	2	200	o	o
Aggregation	2	300	o	-
Grep	2	400	o	-

Table 2: Benchmarks on Private Cluster

thus data locality does not cause a significant performance difference. However, we evaluate our private cluster with the unrealistically large available network bandwidth too, in addition to the constrained 100Mbps switch.

We use Xen 4.0.1 to virtualize physical machines, and all the virtual machines can be reconfigured with the Xen credit scheduler[7], which supports virtual CPU hot-plugging. We implemented the features of dynamic resource reconfiguration on Hadoop 0.20.2.

The two environments, the EC2 cluster and private cluster, have different map and reduce slot configurations considering their core resources. A VM in the Amazon EC2 cluster has 5 map and 3 reduce slots while a VM in the private cluster has 2 map and 1 reduce slots. The input block size of both environments is 128MB, which is more commonly used than 64MB in the Hadoop default configuration. We modified the Hadoop Fair Scheduler [19] for the synchronous and queue-based DRR implementations.

The workloads we use with the EC2 cluster are described in the Table 1. Most of the workloads are from the Hive performance benchmark[5]. Among the workloads, "Grep" and "Select" are I/O intensive workloads, and "Aggregation" and "Join" are communication-intensive workloads. In addition, "Inverted Index" has been added as one of highly communication-intensive workloads [11]. We use a mixture of these workloads, and generate a random submission distribution similar to Zaharia et al [19], which is based on the Facebook trace. Workloads used in the private cluster are described on the Table 2. They are also from the same set of the benchmark, but they are scaled down to fit the resource capabilities of the small cluster.

4.2 Pseudo-ideal DRR Performance

In this section, we first evaluate the potential performance and locality improvements with DRR, by evaluating a single virtual cluster with additional available cores in each physical system. As additional cores are available in each system, a reconfiguration request to add a core to a VM is processed immediately. However, the total number of cores used by the cluster does not change, to assess the performance impact of DRR. To measure this pseudo-ideal DRR performance, we use both of the EC cluster and our private cluster. However, the additionally available cores are limited, so the performance gain can be slightly lower than that with the ideal configuration.

We use the synchronous DRR algorithm for the experiments in this section. A non-local VM (source VM) deallocates a core, while a local VM (target VM) allocates a core to handle an incoming task.

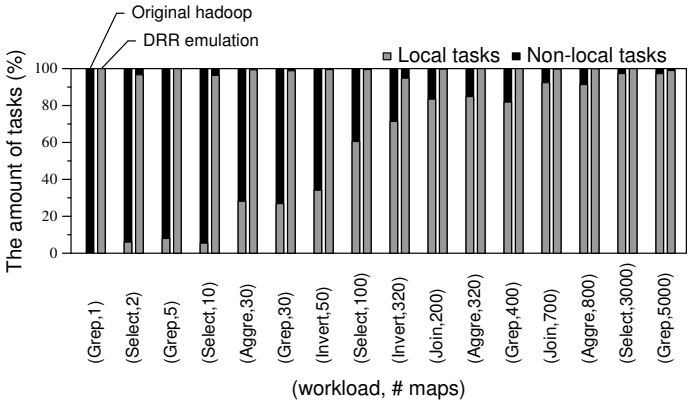

Figure 5: Locality improvements on the EC2 cluster

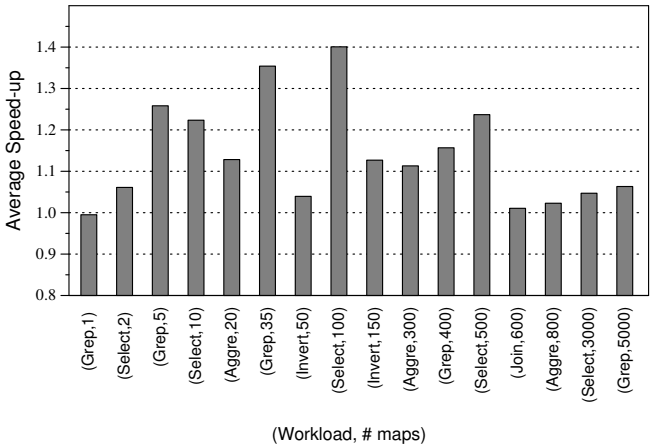

Figure 6: Speedups on the EC2 cluster: normalized to the default Hadoop

4.2.1 *Large-scale Evaluation on the EC2 cluster*

As modifying the hypervisor, or running DRR components at the privileged level in the EC2 cluster is impossible, we emulate the impact of DRR in the EC2 cluster. Firstly, we use a VM instance with eight cores, and initially turned off two CPUs from the VMs. We use the two additional cores to emulate VM reconfigurations to add a core or two to each VM. The total core count for the cluster is fixed to 600, and if an additional core is added to a node, another node must stop using a core to make the total core count fixed to 600. By emulating DRR in this way, we evaluate the performance improvement of the Hadoop cluster using the fixed total number of virtual cores of 600.

Figure 5 shows the locality improvement with the DRR emulation in the EC2 cluster over the plain Hadoop cluster. From the smallest to the biggest job, the data locality of the plain Hadoop jobs increases, because the input blocks of large jobs are broadly spread across all the nodes in the cluster. However, with the DRR emulation, almost all the tasks are executed at the local VMs with their input data. The results do not show 100% data locality with the DRR emulation, since infrequently, two additional cores are not enough for reconfiguration, making this emulation pseudo-ideal.

Due to the locality improvement, the Hadoop cluster with DRR has a significant performance improvement. Figure 6

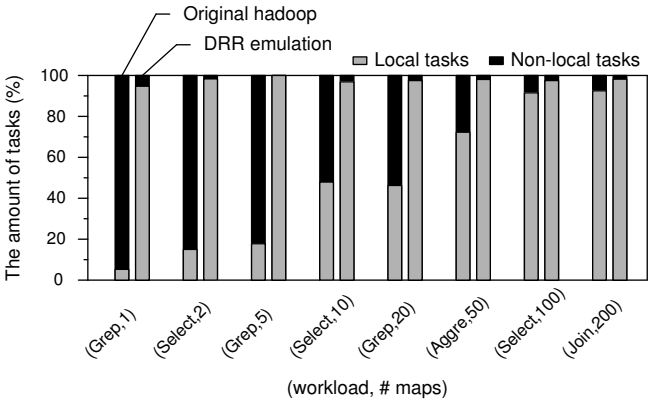

Figure 7: Locality improvements on the 100Mbps private cluster

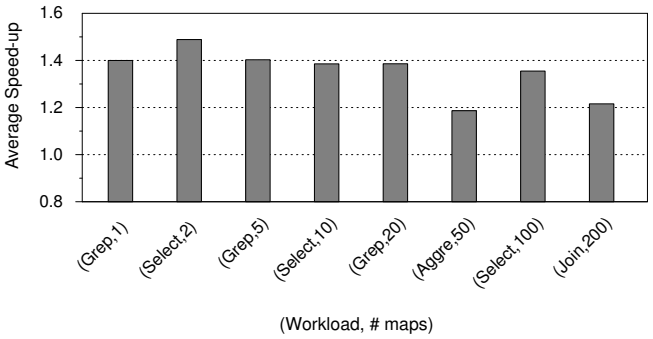

Figure 8: Speedups of synchronous DRR on the 100Mbps private cluster

shows the speedup of DRR on the EC2 cluster, compared to the default Hadoop scheduler. The I/O intensive workloads such as grep or select gain large performance benefits from DRR, but the inverted index workloads do not exhibit performance improvement since almost the entire run-time is spent during the reduce phase, which does not benefit from data locality. It is because this type of jobs needs many reduce tasks to compute a large amount of intermediate output data, whose size is more than three times of the map input data size. On average, DRR can potentially achieve nearly 15% performance improvement over the plain Hadoop.

The extra overheads for selecting a task and a VM for the task to implement DRR by the scheduler are negligible. We measured the scheduling overhead of task scheduling from DRR at the 100-node EC2 cluster. DRR adds about an extra 1 millisecond on average for scheduling, compared to the default Hadoop scheduler.

4.2.2 Evaluation on the Private Cluster

With the private cluster, we evaluate the pseudo-ideal DRR both with 1Gbps switch and 100Mbps switches. For these private cluster experiments, we use 2 vCPUs for each virtual machine initially, and each VM can use additional cores, if locality requires more cores for a VM. Figure 7 shows the locality difference between the base Hadoop and DRR on the private cluster. DRR allows the majority of tasks to run locally, although there are a small portion of non-local tasks, since additional core resources are limited.

Figure 8 shows the performance improvement with the

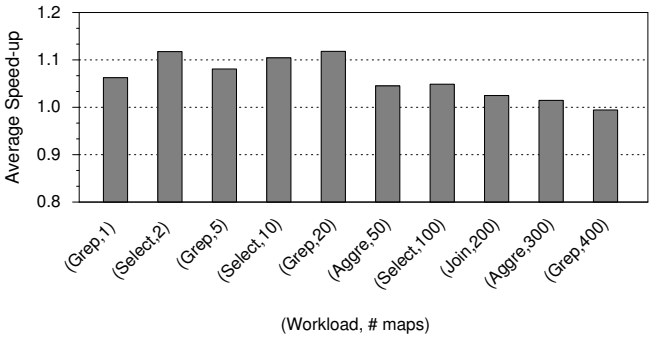

Figure 9: Speedups of synchronous DRR on the 1Gbps private cluster

100Mbps switch. The overall performance improvement is 41% compared to the plain Hadoop on average. Figure 9 presents the performance improvement by DRR with the 1Gbps switch. Since a 1Gbps switch provides very high bandwidth for 6 systems, the potential performance improvements by DRR are much smaller than those with a 100Mbps switch, with about 5% improvement on average.

In this section, we showed the potential performance improvements by a pseudo-ideal synchronous DRR, when one or two additional cores are always available for each VM for reconfiguration. In the next section, we show a realistic setup, where virtual clusters use all the available cores without any headroom, to assess the true benefit of the queue-based DRR.

4.3 Queue-based DRR

In this section, we evaluate the queue-based DRR on the private cluster configuration, with one or more virtual clusters. We first show the performance improvement by the queue-based DRR, and evaluate how much delay the queue-based DRR incurs for the tasks pending in the queue. Since the quest-based DRR implementation cannot run in the EC2 cluster, we use only the private cluster for the experiments. In this setup, all the physical cores are assigned to virtual cores used by virtual clusters, without any free cores available for reconfiguration. Allocation of a core to a VM can occur only when there is a pending deallocation request for a core from the same system.

Figure 10 presents the performance improvement by the queue-based DRR with a 100Mbps switch. Compared to the ideal improvement shown in Figure 8, there is some reduction of performance improvement, since this realistic configuration cannot always provide additional cores immediately, unlike the pseudo-ideal runs. However, the performance improvement is still significant with 35% improvement on average, compared to 41% in the ideal setup.

We evaluate the queue delay of DRR with and without the queue-aware delay reduction schemes discussed in Section 3.3. Figure 11 presents the cumulative distributions of delays without and with the optimization schemes. On both cases, about 50% of allocation requests have zero delay, because there already exists a de-allocation request on the target MRM. However, when we do not apply the queue-aware schemes, a large portion of requests exhibit delays longer than tens of seconds, while the average task execution times are mostly less than 1 or 2 minutes. This is because allocation or de-allocation requests are not been widely spread to

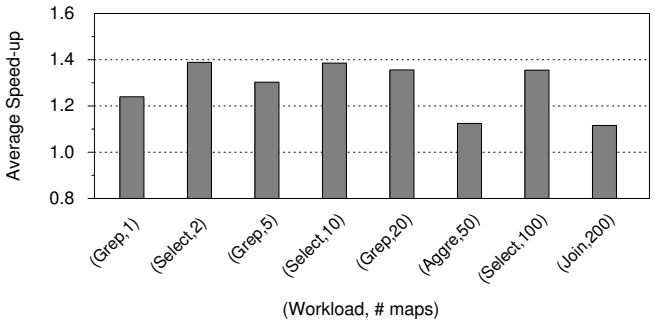

Figure 10: Speedups of queue-based DRR on the 100Mbps private cluster

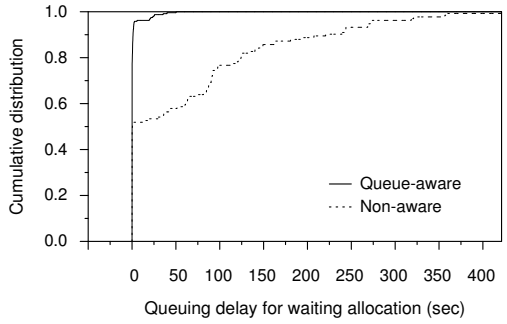

Figure 11: Queue delay distributions with or without queue-aware scheduling schemes

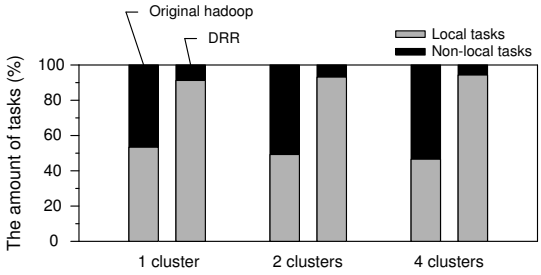

Figure 12: Locality improvements with multiple clusters

virtual machines, and some requests wait for a very long time at the allocation queue due to the skewed scheduling. However, the queue-aware schemes discussed in Section 3.3 for the queue-based DRR reduces the delay significantly, close to zero delay for more than 95% of reconfiguration requests as shown in Figure 11.

Multiple Cluster Evaluation: Multiple clusters can share physical systems and DRR mechanisms, improving the overall utilization of the systems. To evaluate the impact of running multiple virtual clusters, we run 1, 2, and 4 clusters composed of 30, 15, and 7 virtual machines respectively. With the three cluster configurations, we measure the locality improvement by the queue-based DRR. Figure 12 shows the locality improvements with different numbers of clusters sharing the physical cluster. The figure shows that regardless of the number of clusters, DRR can achieve similar locality improvements. Figure 13 presents the speedups of queue-based DRR with the three different numbers of clusters sharing the physical cluster, showing the effectiveness of DRR even with multiple virtual clusters on the same cloud.

Figure 13: Speedups with Multiple Clusters

The results indicate that core exchanges by MRM across different virtual clusters work effectively.

5. RELATED WORK

Improving data locality, while providing fairness among jobs, has been critical for the performance of distributed data-intensive platforms. However, the two goals, locality and fairness, often conflict with each other in distributed platforms. As shown in Section 3.1, a fair scheduling can force a task to be scheduled to a remote node without its data. *Quincy* addressed such data locality and fairness problems on distributed data-intensive platforms by scheduling tasks for fine-grained resource sharing with graph-based workflow models [13]. *Delay Scheduling* proposed locality-aware scheduling policies to enhance data locality for Hadoop platforms [19]. The scheduler delays assigning a task to a node for a short period time until a node containing the input data for the task becomes free. The paper showed that the delays are relatively short in large scale Hadoop platforms, with negligible impacts on fairness.

Several prior studies have been improving MapReduce platforms on virtualized cluster environments. *Purlieus* improved the locality of map and reduce tasks in MapReduce platforms on the cloud by locality-aware VM placement [17]. The authors proposed that exploiting prior knowledge about the characteristics of MapReduce workloads before customers execute them, the cloud scheduler can place data to the proper physical machines using the workload information. Using the data layout, the VM scheduler places VMs to the physical systems with the corresponding input data. The approach differs from DRR, as DRR relies on dynamic VM reconfiguration without any prior information about the workloads.

Sandholm et al. proposed a dynamic VM reconfiguration mechanism with resource hot-plugging, to address skewed resource usages in MapReduce task executions. They assumed that multiple MapReduce clusters share physical resources, and a cluster can use more resources than another cluster, which may violate SLAs. In addition, resource sharing between virtual clusters should occur only within a single physical machine boundary. Kang et al. improved the performance of virtual MapReduce cluster by modifying the context-switching mechanism of the Xen credit scheduler for MapReduce platforms [14]. Zaharia et al. addressed the performance heterogeneity problem of MapReduce platforms on virtual clusters, where sharing I/O and network resources among customers cause performance interferences [20]. Using Amazon Web Service(AWS) such as EC2, S3, SimpleDB, and SQS, Liu and Orban proposed a new MapReduce computation model based the Hadoop platform [15].

There have been several recent studies to manage virtual resources efficiently in cloud systems [8]. Distributed

Resource Scheduler (DRS) proposed a cloud-scale resource management system [6]. The scheduler, from a large resource pool of physical systems, places VMs to maximize the utilization, and live-migrates VMs across physical systems to avoid resource conflicts in a system.

6. CONCLUSIONS

In this paper, we proposed and evaluated a dynamic VM reconfiguration mechanism for distributed data-intensive platforms on virtualized cloud environments, called Dynamic Resource Reconfiguration (DRR). DRR improves the input data locality of a virtual MapReduce cluster, by temporarily increasing cores to VMs to run local tasks. DRR schedules tasks based on data locality, and adjust the computational capability of the virtual nodes to accommodate the scheduled tasks. This approach differs from prior approaches assuming a cluster which always has a fixed amount of computational resource in each node. Using dynamic VM reconfiguration for distributed data-intensive platforms, can be extended to different types of load imbalance. Different resource requirements by different tasks or jobs may cause each virtual node to under-utilize its resource. With VM reconfiguration, each node can be adjusted to provide only the necessary amount of resource demanded for the node. Such a generalized framework with dynamic VM reconfiguration will be our future work. Such a generalized VM reconfiguration framework can lead to customer-friendly configuration methods for cloud resources. Each user may not need to fine-tune the configuration of each virtual machine, as the VM reconfiguration can adjust the individual VM configuration dynamically.

Acknowledgments

This work was partly supported by the IT R&D Program of MKE/KEIT [KI002090, Development of Technology Base for Trustworthy Computing]. It is also supported by the SW Computing R&D Program of KEIT(2011-10041313, UX-oriented Mobile SW Platform) funded by the Ministry of Knowledge Economy.

7. REFERENCES

[1] Amazon Elastic Compute Cloud (EC2). `http://aws.amazon.com/ec2/`.

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

[3] Apache Hive. `http://hadoop.apache.org/hive`.

[4] Apache Pig. `http://pig.apache.org`.

[5] Hive Performance Benchmarks. `https://issues.apache.org/jira/browse/HIVE-396`.

[6] Resource management with VMware DRS. `http://www.vmware.com/pdf/vmware_drs_wp.pdf`.

[7] Xen credit scheduler. `http://wiki.xen.org/wiki/Credit_Scheduler`.

[8] I. A. Ajay Gulati, Ganesha Shanmuganathan. Cloud scale resource management: Challenges and techniques. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[10] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 137–150, 2004.

[11] S. L. Faraz Ahmad and T. V. Mithuna Thottethodi. MapReduce with communication overlap (marco). `http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1412&context=ecetr`, 2007.

[12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd European Conference on Coputer Systems (EuroSys)*, 2007.

[13] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[14] H. Kang, Y. Chen, J. L. Wong, R. Sion, and J. Wu. Enhancement of xen's scheduler for MapReduce workloads. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC)*, 2011.

[15] H. Liu and D. Orban. Cloud MapReduce: A MapReduce implementation on top of a cloud operating system. In *Proceedings of the 11th International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2011.

[16] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 international conference on Management of data (SIGMOD)*, 2010.

[17] B. Palanisamy, A. Singh, L. Liu, and B. Jain. Purlieus: locality-aware resource allocation for MapReduce in a cloud. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[18] C. A. Waldspurger. Memory resource management in vmware esx server. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[19] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer systems (EuroSys)*, 2010.

[20] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.