2009 Eighth International Conference on Grid and Cooperative Computing

XConveryer: Guarantee Hadoop Throughput via Lightweight OS-level Virtualization

An Qin*†, Dandan Tu *†, Chengchun Shu*, Chang Gao‡

*Institute of Computing Technology, Chinese Academy of Sciences, Beijing China
†Graduate University of Chinese Academy of Sciences, Beijing China
‡The Middle School Attached to Northern Jiaotong University
{qinan, tudandan, shuchengchun}@software.ict.ac.cn
gaochang@beihua.edu.cn

***Abstract*—Large-scale data parallel applications such as web indexing, data mining demand plenty of computing and storage resources. As a widely adopted solution, hadoop partitions and distributes the large datasets into chunks across multiple nodes in clusters to process in parallel. For a single cluster node, typically there are several concurrently running applications, sharing and competing system CPU, memory, disk and network bandwidth. The competition will cause unfairness for some jobs and extend their response time. Particularly, some jobs have to be cancelled and rescheduled because of resource starvation. In this paper, we present a solution to reduce resource challenge. The XConveyer, as one of key component, is designed based on kernel-level virtualization facilities, to isolate Hadoop jobs and to guarantee their resource share. Built on XConveyer, Hadoop can enforce scheduling polities down to Operating System, that more fine-granular control can be achieved. The evaluation shows the XConveyer can encapsulate Hadoop jobs executed in isolated containers and guarantee strict control their system resources usages.**

I. Introduction

With data collection growing up to Terabytes level, more and more corporations face the challenge of insufficiency of computing and storage resources. Google [21] presents a simple solution for the data-intensive computing via MapReduce [2] and its GFS [3], however other corporations are not easy to make use of Google techniques because of heavy memory and many restriction, until Hadoop [1] comes into being.

There are two challenges in face of data-process corporation. The first challenge is that more and more data needed to be shared and processed. In astronomy, NASA has collected massive data from about 20 data centers and facts the stress of processing 20TB/day. Also in Energy, the data capability of NERSC has grown with an average quarterly rate of about 29% and will pass the Petabyte point if this trend continues until 2011 [19]. The same case occurred in Google and Baidu. Google grabbles web data from its word and handles about 20 PB in each day. Even in China, online companies, such as Baidu, also need to process the 150TB/day data from several data centers. Coupling with the growth of Cloud market [4], another trend is that companies are apt to pay for the usage of computing and storage resources from Platform providers such as Amazon [22], so as to reduce the cost of hardware and maintaining. Since the distributed infrastructure for large-scale data processing has high complexity and cost in construction and management, it is too expensive for some companies (especially small corporations and personal) to build and own themselves. For example, Google has to pay up to $240m in 2006 for power and space to run its search engine infrastructure. The cost limits the poor companies to utilize Google-like technologies.

Hadoop, built by Yahoo [23], provides a simple Google-like solution for large-scale data processing [1]. It is open-source and now has been widely used in industry and research. Many companies install a unified Hadoop infrastructure to handle data-intensive jobs from multiple internal departments. For example, Baidu [24] uses the Hadoop platform to consume jobs for log processing, web-site statistics, keyword calculation, and so on; alibaba [25] makes use of Hadoop to deal with their departments' jobs such as user behaviors analysis and business log.

It is very common that multiple jobs on behalf of several departments in companies are run on the same hadoop infrastructure. However, multiple departments' jobs running on the same platform may cause resource challenge which may further result in performance overhead and time consumption since defeated jobs have to be rescheduled. Statistical data from our Hadoop platform (holding 10 nodes, 10TB data in HDFS) show several jobs have to be killed and rescheduled and the killed jobs speed up with job number (Figure 1). On the other hand, the resource utilization (shown in Figure 2 by ganglia) is not full when the phenomenon come into being. Also, the total consumed time is extended 3 times. For those departments whose jobs have been waiting, the extended time is expensive since their jobs have same priority and may run in parallel.

In this paper, we present XConveyer, a resource manager for I/O bandwidth enforcement, to reduce job resource challenge in Hadoop based on kernel *cgroups* [26] and *namespace* mechanisms [27]. The *cgroups* mechanism in kernel is exploited to manage resource allocation and scheduling after job has been run as processes in Operating System. Furthermore, kernel namespace mechanism is also adopted to isolate process space from malignant challenge. Based on the support from kernel,

978-0-7695-3766-5/09 $25.00 © 2009 IEEE
DOI 10.1109/GCC.2009.62

IEEE computer society

Authorized licensed use limited to: National Kaohsiung University of Applied Sciences. Downloaded on April 11,2010 at 07:05:42 UTC from IEEE Xplore. Restrictions apply.

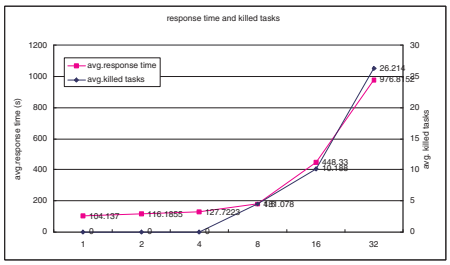

Fig. 1. The killed Hadoop jobs speed up when challenging job number increases

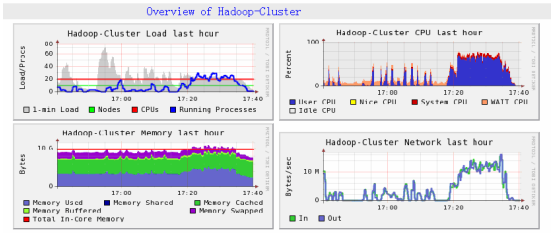

Fig. 2. The 32 jobs have been executed in parallel, and totally consumed 114min. Only in 20min, consumed CPU can high than 50% and consumed memory can touch 50% and network bandwidth is consumed no higher than 10M averagely

the Hadoop is patched to guide job running and scheduling under the control of kernel process group to reduce the challenge on resources.

The rest of the paper is organized as follows: we first describes the background of several technologies related to our methods including OS-level process isolation and enforcement mechanisms in section II. Hadoop's job scheduling is also presented in section II. Then, we detail our techniques for confining job running by grouping hadoop tasks, which also guarantee scheduling policies of hadoop down to kernel to efficiently utilize system resources. The elementary evaluation of our method is given in section IV. Finally, We presents related work in section V and conclude in section VI.

II. Background

In this section, we present related techniques that motivate us to implement the performance enforcement and guarantee of Hadoop performance, including OS-level virtualization facilities such as *Cgroups* and *Namespace*, and job scheduling techniques used by Hadoop.

A. Cgroup mechanism for resource management in kernel

Control groups (*cgroups* for short) is a new mechanism provided in Linux kernel 2.6 as virtualization facilities, that aggregates or partitions sets of tasks and their children into hierarchical groups with specialized behaviors. At user level, the details of task grouping such as CPU, memory and network bandwidth can be manipulated and controlled through a psuedo-filesystem interfaces. At kernel-level, the customized behaviors of task groups can be extended in the form of *subsystems*. Typically, a *subystem* is a resource controller, but it may be anything that control the group of processes, for instance a virtualization system. Compared with other container techniques, *cgroups* is featured with multiple hierarchy support and a lightweight implementation by adding a few simple hooks to the non-critical performance path at OS kernel. Subsystems are suitable to be used for collecting and controlling resource usages. By embedding in specific control codes in these kernel hooks, subsystems are able to penetrate and take control of any particular behaviors of task processes.

There have been several successful applications of *cgroups* for resource control. An expanded CFQ with *cgroups* to control disk I/O operations can achieve more granular and higher throughput [30] [29]. And, *cgroups*-based I/O controller can control the I/O bandwidth in proportional weight for each task group [6]. CPU accessing can be controlled via *cgroups* mechanism, in which process in specific task group will be checked further when CPU scheduler is activated.

B. Namespace mechanism to separate process context in kernel

Originally namespace was a technique developed to create sets of tasks in purpose of migration between hosts. In a namespace the created tasks have a separate space of process IDs, as if they are running on another standalone machine. Currently beside process ID namespace, there are UTS namespace, IPC namespace, NET namespace and MNT namespace in kernel mainstream. The namespaces are created when a new process is cloned and then are attached to the new process. OS kernel differentiates the tasks in two namespaces by comparing both process IDs and task contexts.

Namespace mechanism makes an interesting scenario: two tasks are deemed different from each other even though they have the same *UID*. The technique is useful in isolating processes or tasks for the purpose of resource management in both virtualization and security systems.

C. Job scheduling in Hadoop

Hadoop is a wide adopted implementation of Map-Reduce programming model for large-scale data processing upon clusters. Firstly, Hadoop's scheduler partitions the job into several subjobs (i.e. *tasks* in Hadoop) according to split chunks of data sets. Then Hadoop *maps* the subjobs to the worker nodes, i.e. schedules and dispatches the subjobs to several worker nodes for execution. Lastly, Hadoop *reduces* the results of subjobs, i.e. collects the results from all worker nodes and concludes the final output.

There are two kinds of trackers in Hadoop: one is *JobTracker* which splits dataset into chunks, generates and queues the jobs for chunks; the other is *TaskTracker* which retrieves, runs the tasks and reports the results. When several jobs are submitted, the split jobs may not obtain resources immediately. On the contrary, they are queued by *JobTracker*, and the queued tasks finally are dispatched to worker nodes according scheduling algorithms. It is worthy to notice here that each *TaskTracker* may receive the scheduled tasks in the order

Authorized licensed use limited to: National Kaohsiung University of Applied Sciences. Downloaded on April 11,2010 at 07:05:42 UTC from IEEE Xplore. Restrictions apply.

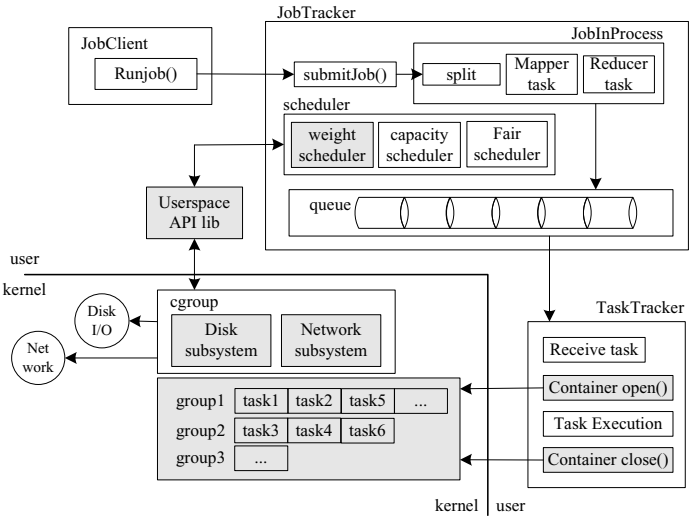

Fig. 3. System Architecture to collaborate Hadoop and *cgroups*

different from the queue order because of the unordered scheduling algorithms and the rescheduling for failed tasks.

The scheduler in Hadoop does not consider what relationship of requests on accessing resource. The *JobTracker* just sends tasks to *TaskTracker* and *TaskTracker* just runs JVM processes to execute tasks and reports corresponding results. If multiple tasks, for example, contend for disk I/O on the same node, nothing can be done in scheduler even though it can foresee the challenge.

III. Design and implementation

In this section, we will present our method in details. Firstly an overview of the method is given, then the *cgroups*-based extensions and user-level functionality are discussed. Furthermore, the details of our weight-based time sharing scheduling algorithms for Hadoop are presented.

A. Overview

The overall architecture which Hadoop is armed with kernel functionalities is shown in Figure 3. The shadowed modules are the extensions for our design, which including new scheduler in *JobTracker*, hooked task execution in *TaskTracker* and the user-level and kernel level *cgroups* control modules. In our extended Hadoop, When a job is submitted by Job client to *JobTracker*, the *JobTracker* first initiates one *JobInProcess* instance which further splits the job into many *Mapper* tasks and *Reducer* tasks according the submitting information (e.g. number of blocks, node busy information, etc.). Tasks are then put into a queue and further scheduled to worker nodes. *TaskTracker* in each worker nodes receives the tasks via periodical heartbeat callback. The received tasks are executed locally. We add hooks in *TaskTracker* to monitor and control task execution. The hooks are invoked before the tracker run tasks, and the task and its children will be managed by *cgroups*-built container via registering their *pid*s. Correspondingly, the *cgroups* extension at OS kernel maintains several hierarchical task chains, managing a group of Hadoop tasks separately. The *cgroups* extension is implemented as a *subsystem* plugged in OS kernel. The subsystem can access specific information and consequently enforce the customized policies for the scheduling of system resources such as CPU, memory and network bandwith. In figure 3, disk I/O and network subsystems are presented, both of which are developed to collect enough system information for each job group and the information is involved in specific schedulers to control particular behaviors of tasks. When a task completes, it is removed from the task chain and clean up in the close hook of *TaskTracker*.

The *JobTracker* scheduler interacts with kernel *cgroups* subsystem through a set of specific API in userspace. The API are designed as channel to control scheduling policy enforcement in OS kernel and the scheduling feedback information including the statistics and accounting data.

B. Cgroup subsystems in kernel

As in Figure 3, two subsystems have been plugged in kernel to control and schedule the system resources.

One is the Disk I/O *cgroups* subsystem that controls allocation of read/write bandwidth according to specific policy. As in [30], the subsystem hooks the bandwidth request at block device level. Whereas our implementation is more intelligent in bandwidth enforcement: processes in a given task group share the bandwidth by configurable weight. Furthermore, our implementation limits the usage for disk space by enforcing the quotas such as the total amount allocation for *inode* and *block*. Figure 4 illustrates the relationship and disk *cgroups* subsystem and how it controls disk resource allocation and scheduling.(??? More details)

The network *cgroups* subsystem is similar as the disk, but it is specialized in controlling the network package in socket.

Authorized licensed use limited to: National Kaohsiung University of Applied Sciences. Downloaded on April 11,2010 at 07:05:42 UTC from IEEE Xplore. Restrictions apply.

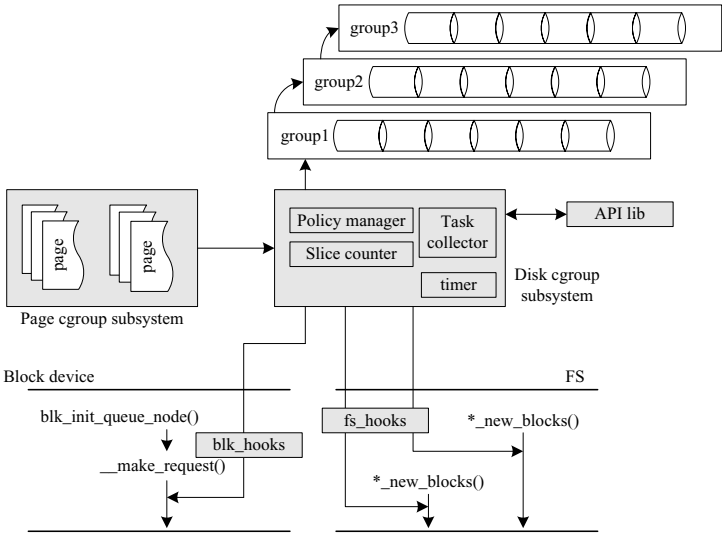

Fig. 4. The *cgroups* subsystems

C. Userspace API library

Similar as *proc* filesystem, *cgroups* facilities also provide Pseduo filesystem (*cgroupfs*, exactly) as default interface between kernel-space and user-space. Generally, the application must obtain root privilege before writing *cgroupfs* to access kernel-space *cgroups* subystems. This limitation is very restrictive for some non-privileged applications including Hadoop.

The solution in *cgroups* API is that the library creates a local Unix socket as communication channel. For security reason, permission checking is enforced before using the communication tunnel. The *cgroups* API contains interfaces in the following:

- Create/destroy, mount/umount *groupfs*, which majors in the operation of *groupfs*;
- Build/destroy a chain for managing task, which can presents kernel information for group chain;
- Add/remove task from existing group, which is used to manage task and group;
- Suspend/resume given task group, which is a set of operation about group chain.

D. Weight-based time sharing scheduler in Hadoop

Since version 0.19, Hadoop has been equipped with flexible framework allowing customized scheduling algorithm to be easily plugged in system. Currently, two scheduling algorithms are distributed with Hadoop release: Fair Scheduling (FS) and Capacity Scheduling (CS). FS scheduling is borrowed from CPU scheduling, but it can only send a group of tasks to same *TaskTracker* and can not suspend them when the assigned time slice is used up [31]. In this sense, FS scheduling does not schedule in a absolutely fair way. CS scheduler is another algorithm of slice allocation. Unlike FS scheduling, CS scheduler classifies the tasks into two categories and put them into two different queues:running and blocked queues. In CS scheduler, it needs to pre-specify a capacity value for each job and a user's limits through the configuration file. The capacity value specifies how many tasks are sent to *TaskTracker* in a given time interval. If the amount of tasks exceeds the value, the scheduler adjusts the value for next time. If the amount exceeds the user's limit, the new jobs are be blocked and put into blocked queue [32].

Both scheduling algorithms want to achieve the efficient control of job on resource allocation and sharing. However, it is hard to guarantee application-level policies would be carried out into worker's OS without any OS-level support, since Hadoop *JobTracker* can not step in *TaskTracker*'s OS.

Our method utilizes the kernel mechanisms to support finer grained scheduling. Building on CS scheduling algorithm, our method adds new subsystems in OS kernel so that *JobTracker*'s scheduling policies can be enforced into each worker's OS. The scheduling policies are defined in Hadoop's master node and mapped to the worker nodes in MapReduce fashion. Application in user-space use the *cgroups* API library as interface to notify the *cgroups* subsystems to make the policies effect. Meanwhile, the kernel information will be reported via API library so that Job scheduler can determine next task delivery accurately.

IV. Performance Evaluation

Evaluation of whole XConveryer implementation described here is at an early stage. Currently, kernel extension based on *cgroups* and *namesapce* facilities has been setup for disk I/O bandwidth. Also, userspace API has been installed and embedded in Hadoop *TaskTracker*. We also accomplish elementary implementation of *cgroups*-based resource accounting in Hadoop CS scheduler. However, they are not at the point where real applications can be run on to experiment the performance guarantee. A more comprehensive performance study will be done when the whole system are finished. In

Authorized licensed use limited to: National Kaohsiung University of Applied Sciences. Downloaded on April 11,2010 at 07:05:42 UTC from IEEE Xplore. Restrictions apply.

this section, we presents our initial experiment with system resource policy enforcement of disk I/O bandwidth.

We do the evaluation using two FIO configuration files [34], which read/write a single 128MB file in two directories. We run the two FIO tests with the two configuration at the same time and measure I/O bandwidth utilization. The result shows two processes can be limited to given disk I/O bandwidth by specified weight or quota. Table I lists some result measured by FIO. Case 1 illustrates the bandwidth consumption and time cost of reading a 128Mb file when the resources are exclusively used by a single process. In case 2, two processes A and B run concurrently in a single node using the same configuration. The result shows the two processes do not share the resources evenly that process B obtains less resources than process A, which resulted in lower bandwith and longer response time. In case 3, we experiment with time slices policy enforcement by configuring the time slices weight to 1 and 4 for the two processes respectively. Processes A has lower average disk I/O bandwidth, and takes more time to completed the disk I/O request. Case 4 enforces the disk I/O bandwidth policy that process A occupies 30% of the total bandwidth. The result shows that process A has half of the average bandwidth of process B and takes one third of total time slice to complete the I/O request, delaying process B.

TABLE I
COMPARISON OF BANDWIDTH CONTROL WITH DISK SUBSYSTEM

		Size (Mb)	Bandwidth (KB/s)	Consumed time(ms)
case 1	process A	128	359	373835
case 2	process A	128	295	453716
	process B	128	191	700739
case 3	process A	128	194	689012
(A:B = 1:4)	process B	128	270	456578
case 4	process A	128	167	862739
(A:30%)	process B	128	298	448716

The disk I/O subsystem is the first step to implement whole design. Another critical system resources should be enforced is network bandwidth. Like Hadoop, many platforms are allowed multiple applications running in the same time, to compete resources each other without any rules. Preventing malignant challenge between jobs need OS-level support for isolation and enforcement. The preliminary performance study shows our method can isolate the execution of processes (or tasks), thus may guarantee the performance of hadoop jobs.

V. RELATED WORK

In this section, we review the work related to our method, which including resource management, job scheduling, virtualization and cloud computing.

A. Resource management

The mechanism of control groups (i.e. *cgroups*) is first added in kernel 2.6, and many subsystems have been developed by open community for resource control. The most interesting feature of *cgroups* is resource accounting for a set of task, where processes and its future children can be contained and controlled by *cgroups* subsystems. Example subsystems in Linux distribution distributed include CPU accounting and memory quota. The CPU accounting subsystem records amounts of time slices used by processes. When time counts exceed the maximal limitation, the CPU subsystem suspend the given group of tasks. Similarly, memory subsystem monitors and controls the memory consumption. The subsystem has several configurable parameters such as maximal limitation, accessible via *cgroupfs*. BFQ [6] is a more complex subsystem that controls I/O queue scheduling. The subsystem achieves higher throughput by grouping the I/O requests in basis of task group and scheduling the requests with auto-tuned slice allocation.

In contrast, XtreemOS [7] performs the resource management at kernel in forms of OS-level extension [9]. The PAM [11] and NSS [12] modules are standard pluggable frameworks supported in mainstream Linux distributions. XtreemOS plugs customized NSS module that allocates different uid/gid for each grid users and thus isolates user and application in coarse-gained way by OS's local instance. PAM module is used to apply customizable resource policies, exploiting the OS capabilities such as *setrlimit()*, quota of filesystem [35].

B. Job scheduling and resource sharing

Previous work of job scheduling focuses the goal on achieving fine-grained workload balance in cluster and other distributed systems. The work reports resource usage, rather than enforces resource policies at runtime. For instance, LSF [17] report detailed resource usage of a whole cluster, and PBS provides a pluggable framework for adding new scheduling algorithms [16]. Both of them dispatch their jobs (not tasks or subjobs in Hadoop) to some worker nodes and then waits for results returned when they finish. In contrast, Hadoop splits a job to multiple tasks that are executed on different worker nodes, achieving high parallel [1].

The common worker nodes to run a job are deemed as trusted nodes in a cluster. Jobs are scheduled among the cluster nodes, with trusted tunnel built with security services based encryption, for example password in OpenSSH [13] and SSL-based connection in Apache [36]. Some of resources that jobs used are based on global storage, for example VMware infrastructure [14] [15]. Resource sharing lays the way for job scheduling, so some research work has already extended the resource sharing scope to not only the servers in data centers but also to the personal PCs and smart mobile devices. Grid middleware (e.g. GSI [18]) builds a resource nodes using a delegation of X509-based certificate in Grid scope, after users have installed the complex middleware before using the feature [18]. XtreemOS approach is to build the verification of X509-based certificate in a modified OpenSSH to enable larger resources sharing for HPC computing [8] without additional package installation.

C. Virtualization and Cloud computing

Virtual Machine (VM) is a popular solution to resource isolation and sharing. Applications in a VM is isolated from

Authorized licensed use limited to: National Kaohsiung University of Applied Sciences. Downloaded on April 11,2010 at 07:05:42 UTC from IEEE Xplore. Restrictions apply.

those in other VMs and those in the host machine. VM is created in application level and emulates hardware to implement resource management, and all the VMs partition the computing and storage resources in single server. Since most servers can not get full loaded 24 hours a day, the free resources can be partitioned for additional benefits. Isolation is another key application of VM for security issues. The strong isolation among VMs reduce minimally demolition from malignant applications. In our method, kernel-level *cgroups* and namespace are more lightweight than most of VM technologies. Furthermore, the container techniques using *cgroups* and namespace do not suffer the tragedy like attacks of Virtual Machine monitor (VMM) from some applications, that may collapse all VMs.

Virtualization also provides a solution for Platform as a Service (PaaS), one of promising features in Cloud computing. However, VM may incur some extra overhead, though the overhead of VM monitor has been reduced to 5% via paravirtualization (more in full-virtualization) [10]. In some realtime applications, the overhead may not be acceptable.

VI. Conclusion and future work

In distributed environment like Hadoop, typically more than one job are dispatched to the same cluster node, where the jobs share and compete for the system resources such as CPU, memory, Disk I/O and network bandwidth. The competition will cause unfairness for some jobs and extend their response time. In this paper, we present a solution to reduce resource challenge. The XConveyer is developed as key component to guarantee resource usage of each jobs can be enforced into kernel, so as to reduce job competition and eliminate the rescheduling. Based on XConveyer, Hadoop can be on the way to guarantee scalable performance regardless the number of submitted jobs. Our method features with OS kernel resource control and policy enforcement in a lightweigt way. The performance evaluation shows the method can put the tasks executed in isolation and strict control their system resources usages. The evaluation of total implementation is weak because patching and enhancing Hadoop is not accomplished. To achieve efficient job execution in a scalable manner, we still need to exploit scheduling feature in Haoop to further enhance resource utilization and to shorten job response time.

Acknowledgment

The authors give thanks to reviewers and valuable comments from colleagues in ICT. Special thanks give to Zhiwei Xu and Haiyan Yu for their insightful discussion. Thanks also go to the supports by XtreemOS project under European Commision FP6 Contract (No. 033576), Natural Science Foundation of China (Grant No. 60873243, 60603004), 863 Program (Grant No. 2008AA01Z140).

References

[1] A Bialecki, M Cafarella, D Cutting. *Hadoop: a framework for running applications on large clusters built of commodity hardware*. OOMalley - Wiki at http://lucene.apache.org/hadoop

[2] J Dean, S Ghemawa. *MapReduce: Simplified data processing on large clusters*. Communications of the ACM, v.51 n.1, January 2008

[3] S Ghemawat, H Gobioff, ST Leung. *The Google file system*. ACM SIGOPS Operating Systems Review, 2003

[4] Brian Hayes.*Cloud computing* Communications of the ACM, v.51 n.7, July 2008

[5] Stephen Soltesz , Herbert Potzl , Marc E. Fiuczynski , Andy Bavier , Larry Peterson.*Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors*. ACM SIGOPS Operating Systems Review, v.41 n.3, June 2007

[6] Paolo Valente, Fabio Checconi.*High Throughput Disk Scheduling with Deterministic Guarantees on Bandwidth Distribution*. http://algo.ing.unimo.it/people/paolo/disk_sched/bfq-techreport.pdf

[7] C. Morin. *XtreemOS: a grid operating system making your computer ready for participating in virtual organizations*. In Proceedings of ISORC07 (July 2007), vol. 5, pp. 347C368.

[8] An Qin, Haiyan Yu, Chengchun Shu, Bing Xu. *XOS-SSH: A Lightweight User-Centric Tool to Support Remote Execution in Virtual Organizations*.the First USENIX Workshop on Large-Scale Computing (LASCO '08),June 2008.

[9] An Qin, Haiyan Yu, Chengchun Shu, Xiaoqian Yu, Yvon Jegou, Christine Morin. *Operating System-level Virtual Organization Support in XtreemOS*. The 9th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'08), Dec. 2008.

[10] Paul Barham , Boris Dragovic , Keir Fraser , Steven Hand , Tim Harris , Alex Ho , Rolf Neugebauer , Ian Pratt , Andrew Warfield. *Xen and the art of virtualization*. Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-22, 2003, Bolton Landing, NY, USA

[11] V. Samar. *Unified login with pluggable authentication modules (PAM)*. Proceedings of the 3rd ACM conference on Computer and communications security, pages 1C10, 1996.

[12] NSSwitch. *System databases and name service switch*. http://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html.

[13] OPENSSH. www.openssh.org

[14] Building the Virtualized Enterprise with VMware Infrastructure

[15] M Nelson, BH Lim, G Hutchins. *Fast transparent migration for virtual machines*. In Proceedings of the USENIX Annual Technical Conference, page 22 - 25, 2005

[16] D Jackson, B Bode, DM Halstead, R Kendall, Z Lei.The portable batch scheduler and the Maui scheduler on linux clusters. Proceedings, 4th Annual Linux Showcase and Conference, 2000

[17] J.A. Kaplan, M.L. Nelson A Comparison of Queuing, Cluster and Distributed Computing Systems, NASA technical Memorandum 109025, NASA LaRC, October, 1993

[18] R. Butler and D. Engert and I. Foster. *A National-Scale Authentication Infrastructure*.IEEE Computer.2000,33(12),60-66.

[19] NERSC report. http://www.pdsi-scidac.org/events/PDSW08/resources/posters/mokhatrani-NGF-poster_SC08.pdf

[20] HADOOP website.http://hadoop.apache.org/core/

[21] Google company. http://www.google.com

[22] Amazon company. http://www.amazon.com

[23] Yahoo company. http://research.yahoo.com

[24] Baidu company. http://www.baidu.com

[25] Alibaba company. http://www.alibaba.com

[26] Control group (cgroup). http://www.mjmwired.net/kernel/Documentation/cgroups.txts

[27] PID namespaces in the 2.6.24 kernel. http://lwn.net/Articles/259217/

[28] A expanded CFQ scheduler for cgroups. http://lwn.net/Articles/306772/

[29] Variations on fair I/O schedulers. http://lwn.net/Articles/309400/

[30] Another proportional weight IO controller. http://lwn.net/Articles/306175/

[31] The fair scheduler in Hadoop. https://issues.apache.org/jira/browse/HADOOP-3445

[32] Capacity Scheduler in Hadoop. http://hadoop.apache.org/core/docs/r0.19.1/capacity_scheduler.html

[33] https://issues.apache.org/jira/browse/HADOOP-3746

[34] FIO. http://freshmeat.net/projects/fio/

[35] Quota in filesystem. http://users.csc.calpoly.edu/ smead/srproj.pdf

[36] OpenSSL. http://www.openssl.org

Authorized licensed use limited to: National Kaohsiung University of Applied Sciences. Downloaded on April 11,2010 at 07:05:42 UTC from IEEE Xplore. Restrictions apply.