Optimizing Big Data Processing Performance in the Public Cloud: Opportunities and Approaches

Dan Wang
Department of Computing
The Hong Kong Polytechnic University
csdwang@comp.polyu.edu.hk

Jiangchuan Liu
School of Computing Science
Simon Fraser University
jcliu@cs.sfu.ca

***Abstract*—Today's lightening fast data generation from massive sources is calling for efficient *big data* processing, which imposes unprecedented demands on the computing and networking infrastructures. State-of-the-art tools, most notably MapReduce, are generally performed on dedicated server clusters to explore data parallelism. For grass root users or non-computing professionals, the cost for deploying and maintaining a large-scale dedicated server clusters can be prohibitively high, not to mention the technical skills involved. On the other hand, public clouds allow general users to rent *virtual machines* (VMs) and run their applications in a pay-as-you-go manner with ultra-high scalability and yet minimized upfront costs. This new computing paradigm has gained tremendous success in recent years, becoming a highly attractive alternative to dedicated server clusters.**

This article discusses the critical challenges and opportunities when big data meet the public cloud. We identify the key differences between running big data processing in a public cloud and in dedicated server clusters. We then present two important problems for efficient big data processing in the public cloud, resource provisioning, i.e., how to rent VMs and, VM-MapReduce job/task scheduling, i.e., how to run MapReduce after the VMs are constructed. Each of these two questions have a set of problems to solve. We present solution approaches for certain problems, and offer optimized design guidelines for others. Finally, we discuss our implementation experiences.

I. Introduction

Today's lightening fast data generation from massive sources and advanced data analytics have made information mining from *big data* possible. We have witness the success of many big data applications. For example, Amazon uses the massive historical shipment tracking data to recommend goods to the very targeted customers, and Google uses billions of query data to predict the flu trends, which can be one week earlier than National Centers for Disease Control and Prevention (CDC).

Bid data processing, however, imposes unprecedented demands on the underlying computing and networking infrastructures. For input data at the petabyte or even exabyte scale, simply improving the power of individual machines, a.k.a, *scale-up*, is hardly practical. State-of-the-art tools, most notably MapReduce, are generally performed on dedicated server clusters to explore data parallelism. They rely on a large scale of machines that work together, a.k.a, *scale-out*, to process the big data in a divide-and-conquer manner. Nevertheless, as compared to the conventional data processing, MapReduce-like tools are still new in the market and there remain much to improve. In particular, from the performance point of view, MapReduce has been criticized for its inefficiency [1].

Moreover, a growing number of big data applications come from not only the IT industry, but also such other areas as civil, environmental, finance, and health industries, to name but a few. For grass root users or non-computing professionals, the cost of deploying and maintaining a large-scale dedicated server clusters can be prohibitively high, not to mention the technical skills involved. On the other hand, public clouds, emerged also in recent years, allow general users to rent computing and storage resources and run their applications in a pay-as-you-go manner. The massive resources available at a public cloud provider's datacenters offer ultra-high scalability and yet minimized upfront costs for its users. This new computing paradigm has gained tremendous success, becoming a highly attractive alternative to dedicated server cluster.

While there have been a flourish of studies on improving MapReduce performance in dedicated server clusters, the research in the context of public cloud is still in its infancy. A public cloud has very different characteristics as compared to dedicated server clusters. It emphasizes the support for effective resource sharing and elastic pay-as-you-go services for general users, and *machine virtualization* plays a key role. For example, Amazon EC2,[1] the most successful and widely used Infrastructure-as-a-Service (IaaS) cloud platform heavily relies on the Xen virtualization in its deployment. Each virtual machine (VM), known as an *instance*, functions as a virtual private server; a user can rent VMs of different configurations under different prices.

This article discusses the unique challenges and opportunities when big data meet the public cloud. We identify the key differences between running big data processing in a public cloud and in dedicated server clusters. We then present two main questions for efficient big data processing under MapReduce in the public cloud, resource provisioning, i.e., how to rent VMs and VM-MapReduce job/task

*Dan Wang's work is supported in part by National Natural Science Foundation of China (No. 61272464), RGC/GRF PolyU 5264/13E, HK PolyU 1-ZVC2, G-UB72. Dan Wang is the corresponding author.

†J. Liu's research is supported by a Canada NSERC Discovery Grant, an NSERC Strategic Project Grant, and a China NSFC Major Program of International Cooperation Grant (61120106008).

[1] http://aws.amazon.com/cn/ec2/.

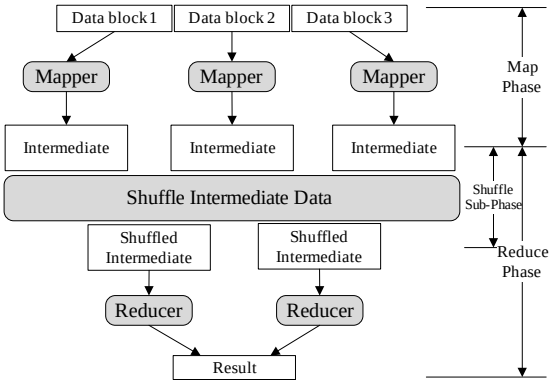

Fig. 1. The MapReduce architecture

scheduling, i.e., how to run MapReduce after the VMs are constructed. We systematically examine the performance of the key problems facing these two questions, including virtual machine job scheduling, shuffling, data locality, as well as resource provisioning under CPU-I/O contentions. We review the potential solutions in addressing some problems and offer optimized design guidelines for others. Finally, we discuss our implementation experiences. We release our codes, scripts and documentation as open sources.

II. When Big Data meet public cloud

Among the many tools that scale out big data processing to parallel machines, MapReduce (proposed by Google), is arguably the most popular, and has become the *de facto* standard nowadays. Fig. 1 illustrates the basic MapReduce structure. A MapReduce job consists of two phases (or tasks), namely, *map* and *reduce*. Accordingly, there are two program functions: *mapper* and *reducer*. In the map phase, the input data is split into blocks. The mappers then scan the data blocks and produce intermediate data. The reduce phase starts from a *shuffle* sub-phase, which, run by the reducers, shuffles intermediate data and moves them to the corresponding nodes for reducing. The reducers then process the shuffled data and generate the final results. For complex problems, multiple rounds of map and reduce can be executed.

There have been many practical MapReduce implementations, and the open source Hadoop[2] is the most widely used to date. There have also been MapReduce-like tools that target specific application scenarios; for example, YARN for multi-stage cluster management with global resource management, Pregel for big graph processing, GraphLab for parallel machine learning, PowerGraph for machine learning on nature graphs, and Spark for distributed in-memory computation.

Today Hadoop or other MapReduce-like tools are generally running on dedicated server clusters. Existing studies thus have mainly focused on optimization with such physical infrastructures. A public cloud however has very different characteristics as compared to dedicated server clusters. Through *machine virtualization*, it effectively hides the many levels of implementation and platform details, making shared resources as if being exclusive to the end users [2]. In such a virtualized environment, user applications are sharing the underlying hardware by running in virtual machines (VMs). Each VM, during its initial creation (by the users), is provisioned with a certain amount of resources (such as CPU, memory and I/O). The number and capacity of the VMs can be requested in a pay-as-you-go fashion, or even adjusted in runtime. In other words, the cloud resource allocation is highly elastic, and an application may adjust the resources for processing its big data at different stages. The performance of such resources as CPUs and memories are relatively the same with both physical or virtual machines. This however becomes no longer true for shared resources such as network bandwidth and I/Os. Even worse, it is known that there are conflicts between networking, I/O and CPU. More specifically, given that virtualization often requires a hypervisor to intercept system calls, when there are intensive networking and I/O operations, it will influence the CPU's operation, thereby affecting its performance [3]. As such, big data processing in the public cloud can differ dramatically from that in dedicated server clusters, and the above factors must be taken into account.

In this virtualized, shared, and highly elastic environment, two cascaded issues are to be addressed:

1) *Resource Provisioning*: Given a MapReduce application, find the best way to construct VMs;
2) *VM-Job Scheduling*: Given the set of VMs that have been provisioned/constructed, find the best way to run MapReduce jobs/tasks.

Obviously, optimized resource provisioning and job scheduling depends on accurate cloud performance measurement and modeling. Modeling cloud performance is a difficult problem. The behavior of many cloud applications, such as game servers and web servers, changes frequently as it closely depends on user behaviors. There is a major difference in big data processing applications, however. A big data application runs with the same type of data again and again. For example, Google flu prediction runs every day. It uses user query data. Though the queries are different from day to day, the amount of data, the amount of data for each data block used in MapReduce and the way that Google flu prediction processes these data are the same. This makes it possible to predict the performance of VMs, mapper tasks, reduce tasks, etc. As such, one can use one round of data to obtain the performance modeling. Such information serves as the input for our resource provisioning and scheduling.

III. Resource Provisioning in the Public Cloud

We first look into the resource provisioning problem. There have been initial studies on running MapReduce in the cloud, addressing the resource provisioning issues [4][5]. The strategies that optimizes the cluster size according to different job types and workloads have also been presented [6]. These pioneer studies consider different VM types as containers with given computing capacities. As discussed, in the public cloud, the labeled capacity may not be accurate, particularly for data intensive applications.

[2]http://hadoop.apache.org/.

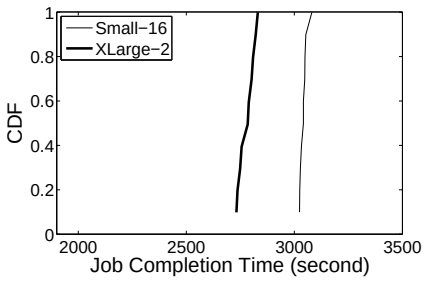

Fig. 2. The CDF of the processing time for the same MapReduce job under different cluster configurations in EC2.

To understand this, we construct two clusters in Amazon EC2: 1) Small-16 with 16 EC2 small instances; 2) XLarge-2 with 2 EC2 extra large instances. Both clusters have identical CPU capacity and memory capacity. We also assume that the unit-time costs for using both clusters are identical. As such, one should expect that their performance to be the same. We show that this is not the case when running MapReduce.

We execute the same MapReduce job on both clusters, and the results are shown in Fig. 2. When two extra large instances are employed in our cluster (referred to as XLarge-2), the job can be done between 2732 to 2831 seconds. When 16 Small instances are used (referred to as Small-16), the job completion time increases to 3042 seconds. In other words, the job runs 10% slower on Small-16 than on XLarge-2, indicating that a user will suffer a 10% capital loss if he does not construct the VMs wisely. This is because, with I/O-intensive operations, there are strong interference between CPU and I/Os.

In [3], the resource provisioning problem in the cloud-based big data systems is re-modeled and an interference-aware solution that smartly allocates the MapReduce jobs to different VMs is presented. The key is the remodeling phase, where the interference is captured into a parameter and is formulated into the performance of mapper, reducer etc. The trace-driven evaluation using the Facebook data and four types of Amazon EC2 instances (small, media, large and extra large) show that, as compared with a state-of-the-art cost-aware resource provisioning algorithm [7], the interference-aware solution outperforms by 11%.

IV. MapReduce VM-Job Scheduling

We next look into VM-job scheduling, assuming that the set of VMs have already been constructed. We first discuss how to schedule the MapReduce jobs, tasks and how to assign them on the VMs. We then look into the shuffling sub-phase and examine how online data-machine assignment can be optimized. Finally, we discuss an notion of data locality and offer design guidelines through cloud elasticity.

A. VM-Job Scheduling

In the default MapReduce implementation, each machine is assigned two mappers and one reducer, and multiple jobs are randomly assigned to the machines. Consider an example in Fig. 3. There are three VMs and two jobs. Job 1 has 4 map tasks and 2 reduce task. Job 2 has 1 map task and 1 reduce task. Assume the processing time to be 75 seconds for all map tasks and 100 seconds for all reduce tasks. If the VM assignment is not considered, we will have Fig. 3(a), which follows the default FIFO strategy of Hadoop. If we jointly consider VM assignment, we can achieve a schedule in Fig. 3(b). It is easy to see that the completion time of job 2 in Fig. 3(a) is 250 seconds, whereas in Fig. 3(b), it becomes 175 seconds, a 30% improvement.

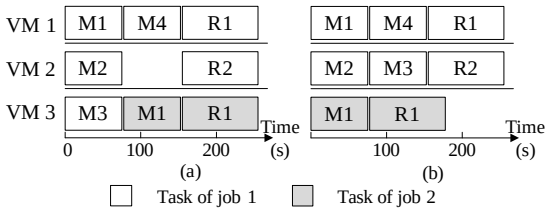

Fig. 3. VM-Job/task scheduling, (a) without taking VM assignment into consideration, (b) taking VM assignment into consideration.

The above example shows that the simple default setting is hardly optimal. This is a scheduling problem, yet MapReduce has a unique parallel-sequential structure that worth special consideration. More formally, we have

MapReduce VM Job Scheduling (MVJS): Given a set of MapReduce jobs with map tasks and reduce tasks (map tasks need to be in precedence of reduce tasks), a set of VMs with certain performance, find a feasible schedule (when and where to run the tasks) to minimize total job completion time.

There are a series of works to progressively investigate this problem. In [8], a scheduling algorithm OffA of multiple MapReduce jobs on multiple machines is developed. In [9], the study is extended where the parallel-sequential structure is taken into consideration and an 8-approximation algorithm, H-MARES, is developed. Both studies do not consider the machine assignment. The latest episode is [10] where the MapReduce job/task scheduling is jointly considered with machine assignment. A 3-approximation algorithm, MarS, is developed. MarS first relaxes MSJO into a linear-programming problem with a readily available optimal solution. This gives a lower bound to MSJO and sorts out the optimized scheduling order. A greedy search is then conducted for a feasible solution and approximately solves MVJS.

Performance improvement can be observed in each of these different progresses. For example, a gain of 20% of MarS to H-MARES is observed in most typical cases [10].

B. Online Optimal Shuffling

Looking further into the MapReduce structure, there is a shuffling subphase in the reduce task. The map function transfers the input raw data into (key, value) pairs and the reduce function merges all intermediate values associated with the same intermediate key. The shuffling subphase starts after some map tasks finish and runs in parallel with other map tasks. Intrinsically, the shuffling sub-phase will re-assign intermediate data into appropriate VMs to run. In the state-

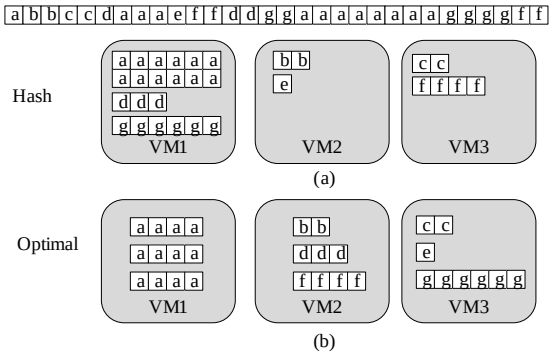

Fig. 4. Shuffle scheduling, (a) default shuffle scheduling, (b) load-aware shuffle scheduling

of-the-art implementations, a hash function is used for the re-assignment, which directly affects the load of different machines. If the data input has a uniform distribution, the load can be expected to be automatically balanced. If the data is skewed, the performance can be poor however.

To illustrate this, consider the example in Fig. 4. Assume that there are three VMs for a typical Wordcount application (to calculate the number of words in a document). If the default hash function is used, the result will be Fig. 4(a). Here the words are distributed into three VMs in a round-robin fashion. As such, VM1 has a, d, g, VM2 has b, d, and VM3 has c, f for further processing. Clearly, the load is unbalanced as the distribution of the words is highly skewed. Fig. 4(b) shows a better distribution, which yields shorter finishing time.

This again shows that there is large room towards an optimal strategy for big data processing. More formally, we have

MapReduce Shuffle Scheduling (MSS): Given a data stream produced by a MapReduce job, a set of VMs, find a partition strategy for the data stream, so as to minimize the maximum load of the VMs.

To minimize the overall finishing time, the shuffle subphase should start as soon as possible. In other words, we need an online algorithm for problem MSS.

From the theoretical point of view, this is related to the *online minimum makespan problem* where jobs come one by one with processing time and they need to be assigned to parallel machines. Yet a unique constraint in the MapReduce context is that the same key must go to the same machine.

A List-based Online Scheduling (LOS) algorithm is developed in [11]. LOS decides, upon receiving a (key, location) pair, to which machine to assign that item without any knowledge of what other items may be received in the future. LOS assigns the unassigned keys to the machine with the smallest load once they come in and for the assigned keys, LOS just follows the constraints that the same key should go to the same machine. It is shown that LOS has a competitive ratio of 2, i.e., it yields an overall finishing time at most twice of the optimal solution.

C. Data Locality and Cloud Elasticity

An important notion in big data processing is *data locality*. Data parallelism scales out the data processing to multiple machines, which incur data movement from one VM to another. Massive data movements, involving slow networking and disk operations, can aggravate resource contention and introduce excessive delay. It is therefore desirable to data close to the target machines.

Wang et al. [12] investigate data locality of map tasks in scheduling under heavy traffic. To balance between data-locality and load-balancing while maximizing throughput and minimizing delay, the system is modeled into a queueing architecture. Then, a scheduling algorithm based on Join the Shortest Queue (JSQ) policy and MaxWeight policy is proposed. It is proven that the proposed algorithm is throughput optimal. Tan et al. [13] propose a stochastic optimization framework to optimize data locality of reduce tasks. Based on the optimal solution under restricted conditions, a receding horizon control policy is proposed.

Note that the public cloud has much greater capacity beyond the capacity requirement of one big data processing application. Together with the elasticity nature of the public cloud, this offers opportunities to change VM capacity during runtime [14] to accommodate different data intensities in different stages of MapReduce. In other words, rather than moving data, it is possible to change the VM capacities at different times, according to the amount of data to be processed in local VMs. In the initial investigation [14], for a certain MapReduce job, and a set of VMs each of which has a set of CPUs, when a task is planned to be executed, the number of CPUs in different VMs is adjusted according to the location of data to be processed by this task. Experiments in real cluster and EC2 cluster show that throughput of Hadoop is improved by 41% and 15%, respectively.

Currently, runtime elastic VMs are not available in existing cloud providers. In Section V, we show our initial implementation experience in constructing runtime elastic VMs where we adjust the number of CPU in each VM. Using Xen hypervisor, such adjustment introduces ignorable delay. With runtime elastic VMs, we observe a 43% earlier in finishing time, where the total number of CPU $\times$ hours remain similar. We believe that there are immense opportunities for both performance optimization and pricing here.

V. Implementation Experiences

So far we have systematically reviewed the approaches toward optimizing MapReduce for big data processing in the public cloud. We now present our implementation experiences of an integrated optimization framework, and our codes, scripts and documentation/manual are available as open sources.[3]

We implement with Hadoop-1.2.0 running in the Amazon EC2 public cloud. In this implementation, we need to coordinate two components within Hadoop, namely, JobTracker and TaskTracker. JobTracker manages all jobs in a Hadoop cluster and, as jobs are split into tasks, TaskTracker is used to manage tasks on every machine. The implementation framework is shown in Fig. 5. We add a new big data

[3]http://www4.comp.polyu.edu.hk/csyiyuan/projects/MarS/MarS.html

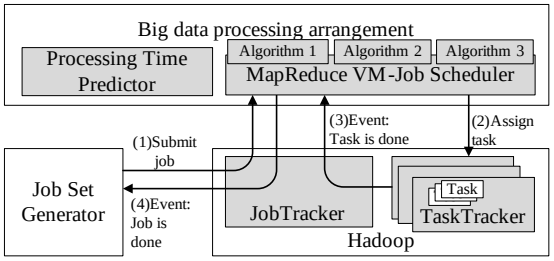

Fig. 5. The implementation framework

processing arrangement component. In this component, there is a predictor module to evaluate the performance of jobs, tasks and VMs. We register a MapReduce VM-Job Scheduler module to JobTracker so that JobTracker can call this module to make scheduling decisions. The MapReduce VM-Job Scheduler module makes decisions according to the algorithms that we develop in this project. We show in Fig. 5 the event driven steps of a Hadoop run time. More specifically, there are four associated steps: 1) when a job is submitted to Hadoop, JobTracker notifies MSJO that a job is added. MSJO puts the job into a queue; 2) MSJO scheduler is event driven from JobTracker. When Hadoop is running, JobTracker keeps notifying MSJO on TaskTracker status and if a machine is idle, MSJO assigns a task to the TaskTracker of this machine; 3) after a task finishes, the TaskTracker informs JobTracker, which will further notify MSJO and MSJO updates job information; and 4) if all tasks in a job finish, MSJO removes the job and JobTracker sends a job-completion event to user application.

As an working example, we evaluate algorithms MarS and H-MARES, discussed in Section IV.A, with experiments on a cluster. This cluster is built with 16 Amazon EC2 small instances. We employ Wordcount, a benchmark application, as the MapReduce program in our experiments. We use a document package from Wikipedia as input data of jobs. This package contains all English documents in Wikipedia since 30th January 2010 with uncompressed size of 43.7 GB. We build a job containing 10 jobs where input data size of every job is less than 1GB. We use this job set to evaluate the performance of our algorithms when jobs are small. In the experiments the total weighted job completion time of H-MARES is 5224 seconds while MarS is 4826 seconds. Such results well match our earlier simulation results.

Currently, we are also developing runtime elastic VMs. We show the benefit in a demo experiment. We employ Wordcount as the MapReduce job and the input data is 12G Wikipedia data. We compare two runtime strategies: static VM strategy and elastic VM strategy. In the static VM strategy, every slave node has 1 CPU. In the elastic VM strategy, every slave node has 2 CPUs at the job starting time and the number of CPUs decreases to 1 after all map tasks finish. In the experiment, the total CPU number $\times$ CPU running time of both strategies are comparable. Yet, the elastic VM strategy finishes the job in 3934 seconds while static VM strategy is 6890 seconds, an impressive improvement of 42.9%.

VI. Conclusion and Future Work

In this article, we discussed the suitable user groups in running big data applications in the public cloud. We showed that, as compared to Google, who run their big data processing applications on dedicated server clusters, the grass-root users and non-computing professionals may only resort to the public cloud. We illustrated the key differences between the public cloud and dedicated server clusters. We discussed two important problems for efficient big data processing in the public cloud. We presented solution approaches for certain problems, and offered optimized design guidelines for others.

Nevertheless, the research and practices of big data processing in the public cloud remains in its infancy. Many differences between the public cloud and dedicated server clusters are left unexplored; and many existing questions are still yet to have clear winning solutions. With the grass-root users and non-computing professionals becoming aware of running big data applications, there will be abundant opportunities. Especially, we consider better understanding on the impact of networking on the VM performance and cloud elasticity may improve or even drastically change the problem spaces in resource provisioning as well as MapReduce job scheduling.

References

[1] D. DeWitt and M. Stonebraker, "Mapreduce: A major step backwards", the Database Column, 2008.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Comm. of the ACM*, vol. 53, pp. 50-58, Apr. 2010.

[3] Y. Yuan, H. Wang, D. Wang, and J. Liu, "On Interference-aware Provisioning for Cloud-based Big Data Processing" in *Proc. ACM/IEEE IWQoS'13*, Montreal, Canada, 2013.

[4] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud", in *Proc. USENIX HotCloud'09*, San Diego, CA, 2009.

[5] F. Tian and K. Chen, "Towards optimal resource provisioning for running mapreduce programs in public clouds," in *Proc. IEEE Cloud'11*, Washington DC, 2011.

[6] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics," in *Proc. ACM SoCC'11*, Cascais, Portugal, 2011.

[7] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud", in *Proc. IEEE ICDCS'11*, Minneapolis, MN, 2011.

[8] H. Chang, M. Kodialam, R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, "Scheduling in mapreduce-like systems for fast completion time," in *Proc. IEEE INFOCOM'11*, Shanghai, China, 2011.

[9] F. Chen, M. Kodialam, and T. Lakshman, "Joint scheduling of processing and shuffle phases in mapreduce systems," in *Proc. IEEE INFOCOM'12*, Orlando, FL, 2012.

[10] Y. Yuan, D. Wang, and J. Liu, "Joint Scheduling of MapReduce Jobs with Servers: Performance Bounds and Experiments" in *Proc. IEEE INFOCOM'14*, Toronto, Canada, 2014.

[11] Y. Le, J. Liu, F. Ergun, and D. Wang, "Online Load Balancing for MapReduce with Skewed Data Input", in *Proc. IEEE INFOCOM'14*, Toronto, Canada, 2014.

[12] W. Wang, K. Zhu, L. Ying, J. Tan and Li Zhang, "Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality", in *Proc. IEEE INFOCOM'13*, Turin, Italy, 2013.

[13] J. Tan, S. Meng, X. Meng and Li Zhang, "Improving ReduceTask Data Locality for Sequential MapReduce Jobs", in *Proc. IEEE INFOCOM'13*, Turin, Italy, 2013.

[14] J. Park, D. Lee, B. Kim, J. Huh, and S. Maeng, "Locality-aware dynamic vm reconfiguration on mapreduce clouds", in *Proc. ACM HPDC'12*, Delft, the Netherland, 2012.