2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Maestro: Replica-Aware Map Scheduling for MapReduce

Shadi Ibrahim*†, Hai Jin*, Lu Lu*, Bingsheng He‡, Gabriel Antoniu†, Song Wu*

*Cluster and Grid Computing Lab
Services Computing Technology and System Lab
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, 430074, China
hjin@hust.edu.cn

†INRIA Rennes - Bretagne Atlantique
Rennes, 35000, France
{shadi.ibrahim, gabriel.antoniu}@inria.fr

‡School of Computer Engineering
Nanyang Technological University
Singapore, 639798
bshe@ntu.edu.sg

***Abstract*—MapReduce has emerged as a leading programming model for data-intensive computing. Many recent research efforts have focused on improving the performance of the distributed frameworks supporting this model. Many optimizations are network-oriented and most of them mainly address the data shuffling stage of MapReduce. Our studies with Hadoop demonstrate that, apart from the shuffling phase, another source of excessive network traffic is the high number of map task executions which process remote data. That leads to an excessive number of useless speculative executions of map tasks and to an unbalanced execution of map tasks across different machines. All these factors produce a noticeable performance degradation. We propose a novel scheduling algorithm for map tasks, named Maestro, to improve the overall performance of the MapReduce computation. Maestro schedules the map tasks in two waves: first, it fills the empty slots of each data node based on the number of hosted map tasks and on the replication scheme for their input data; second, runtime scheduling takes into account the probability of scheduling a map task on a given machine depending on the replicas of the task's input data. These two waves lead to a higher locality in the execution of map tasks and to a more balanced intermediate data distribution for the shuffling phase. In our experiments on a 100-node cluster, Maestro achieves around 95% local map executions, reduces speculative map tasks by 80% and results in an improvement of up to 34% in the execution time.**

***Keywords*-MapReduce; Hadoop; cloud computing; replication; scheduling;**

I. Introduction

Data volumes are ever growing, from traditional applications such as databases and scientific computing to emerging applications like Web 2.0 and online social networks. This has boosted the intensity of recent research on scalable data intensive systems, including Dryad [1] and MapReduce [2]. Among such systems, Hadoop, an open-source MapReduce implementation, has widely been adopted by industries such as Facebook, and academia. Recently, Hadoop has been deployed in many cloud platforms. For example, Amazon has equipped their software stack with Hadoop [3] to facilitate running large-scale data applications on Amazon EC2 [4]. The New York Times rented 100 virtual machines for a day to convert 11 million scanned articles to PDFs [5]. Due to its wide adoption, the performance of Hadoop in particular (and MapReduce in general) has become an important research topic [6–10]. This paper follows this line of research and contributes to the goal of improving the performance of MapReduce frameworks.

MapReduce [2] was originally proposed by Google to simplify development of web search applications on a large number of machines. The MapReduce [2] system runs on top of the *Google File System* (GFS) [11], where files are partitioned into chunks, and each chunk is replicated to tolerate failures. Data processing is co-located with data storage; when a node is available to process a map task, the job scheduler consults the storage metadata service to get the hosted chunks as close as possible, in this order: on the same node, on another node within the same rack and on another node outside the rack. The goal is to leverage data locality. Moreover, data replication makes the MapReduce system not only fault-tolerant, but also facilitates the handling of slow nodes ("stragglers"). For example, MapReduce runs *speculative copies* of in-progress stragglers' tasks on other machines to finish the computation faster.

While this simple algorithm considers data locality for efficiency, it is unaware of the consequences on scheduling the next task in terms of the possibility of local map task execution. For example, consider a situation with two data nodes, A (hosts two chunks $\{r_1, r_2\}$) and B (hosts $\{r_1\}$); at run time, if node A reports an empty task slot, the current Hadoop scheduler will blindly select one chunk without considering the consequences of processing each hosted chunk; Processing r_1 on node A will cause a non-local map task execution on node B. Our experiments demonstrate that approximately 23% of the map tasks are non-local map tasks

978-0-7695-4691-9/12 $26.00 © 2012 IEEE
DOI 10.1109/CCGrid.2012.122

(i.e., the input of the map is on a remote machine).

Clearly, this excessive number of non-local map tasks causes costly network traffic, and thus performance degradation. It also causes performance degradations in other ways:

1) The non-local map tasks may needlessly create speculative map tasks. Because non-local map tasks, due to the data transfer, have longer execution time compared to local map tasks, they have more potential to be categorized as slow tasks (and need to run speculative copies). Our experiments demonstrate that approximately 50% of the speculative map tasks are backups of non-local map tasks, and even worse, 50% of these speculative map tasks are unnecessary.
2) The non-local map tasks along with the unnecessary speculation may lead to unbalanced execution of map tasks across different data nodes because some nodes will be busy executing longer map tasks. This in turn may cause reduce-shuffle skew.

Recognizing that the current Hadoop's scheduler for map tasks causes a high number of non-local map executions, as it disregards replicas distributions, we seek an approach to reduce the non-local map task executions, reduce the unnecessary map speculations, and balance the number of map tasks executions among data nodes (each node executes nearly the same number of map tasks). We have developed a scheduling algorithm – Maestro – to alleviate the non-local map tasks executions problem of MapReduce. Maestro is conducive to improving the locality of map tasks executions efficiency by virtue of the finer-grained replica-aware execution of map tasks, thereby having one additional factor for the chunk hosting status – the expected number of map tasks executions to be launched. Maestro keeps track of the chunks' locations along with their replicas' locations and the number of other chunks hosted by each node. In doing so, Maestro can efficiently schedule the map task to the node with minimal impacts on other nodes' local map tasks executions.

We have evaluated Maestro on 100-nodes of Grid5000 [12, 13]. With the *sort* and *wordcount* benchmarks, we demonstrate that Maestro achieves around 95% local map executions and reduces speculative map tasks by 80%. As a result, it improves the response time of Hadoop by 34% and 22% for *sort* and *wordcount* workloads, respectively.

The rest of this paper is organized as follows. Section 2 discusses the current map task scheduling and the impacts of non-local map task execution through a series of experiments. The design of our Maestro algorithm is presented in section 3, followed by a performance evaluation in section 4. Finally, we review related work in section 5, and conclude the paper in section 6.

II. Map Scheduling in MapReduce

Before we introduce our approach, let us first illustrate the current map tasks scheduling used in Hadoop. We then discuss its shortcomings through a motivating example.

A. Map Task Scheduling in Hadoop

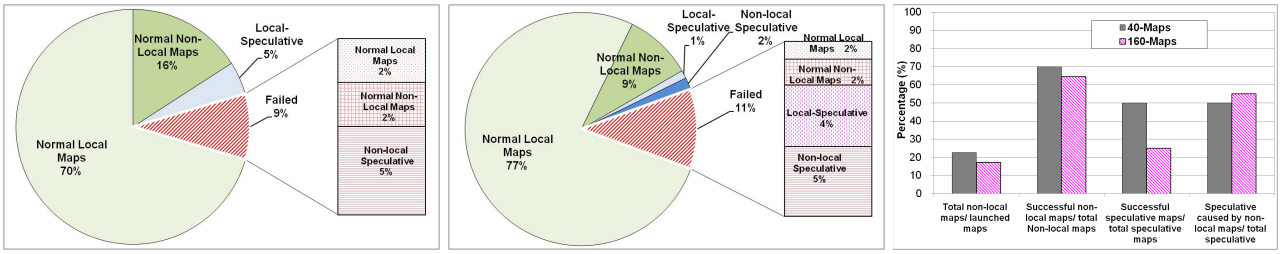
In the current version of Hadoop, the map tasks are scheduled as follows. Initially, the Hadoop master assigns the map tasks to the slaves (depending on the slaves' slots capacity), considering data locality. At run time, when a slave reports an empty slot to process map task, the job scheduler consults a storage metadata service to get the hosted chunks in three successive ways: by that node, by another node within the same rack, and then by another node outside the rack. This order aims to exploit data locality. New map task from the *map tasks pool* will be assigned considering the aforementioned sequence. Moreover, Hadoop-MapReduce handles failures in the following way. If a node crashes, MapReduce re-runs its tasks on a different machine by giving these tasks higher priority in the *map tasks pool* (the map tasks pool has three kinds of map tasks: failed map tasks which have highest priority, normal map tasks and speculative map tasks with lowest priority). Finally, data replication not only makes the MapReduce system fault tolerant, but also helps in handling stragglers: speculative copies of straggler's in-progress tasks are scheduled on other machines to help reducing the overall computation time.

B. A Motivating Example

We perform a simple experiment as a motivating example to illustrate the problem of non-local map execution in Hadoop.

1) Experimental setup: Our experimental hardware consists of a 6-nodes one-rack cluster. Each node is equipped with two 4-core 2.33GHz Xeon processors, 8GB of memory and 1TB of disk, runs RHEL5 with kernel 2.6.22, and is connected with Gigabit Ethernet. We use virtualization to scale the number of nodes in Hadoop using Xen 3.2 [14, 15]. In the virtualized environment, we deploy four *virtual machines* (VMs) on five of these *physical machines* (PMs), reaching a cluster size of with 20 slave. Each VM ia configured with 1 pinned VCPU, 1GB memory and 60GB of virtual disk and is running with REHEL5, kernel 2.6.22. All results described in this paper are obtained using Hadoop version 0.19.0, and the data is stored with 2 replicas per chunk in *Hadoop's Distributed File System* (HDFS). We select the *sort* application from the GridMix benchmark.

2) Results and discussion: Figure 1 shows the experiment results for map tasks executions in *sort* application. We vary the number of map tasks to 40-maps and 160-maps which is equivalent to running *sort* with two data sets 2.5GB and 10GB, respectively. The minimum and average completion time of the local map tasks are (2 and 11) seconds and (2 and 49) seconds for the 40-maps and 160-maps jobs, respectively. In contrast, the minimum and average completion time of the non-local map tasks are (5 and 20) seconds



(a) Map tasks: 40-Maps (b) Map tasks: 160-Maps (c) Job statistics

Figure 1. Impacts of non-local map tasks in *sort*

when running the 40-maps job; and (9 and 60) seconds when running the 160-maps job.

Figures 1 (a) and (b) show the breakdown of map tasks executions for two jobs with 40 and 160 map tasks, respectively. The number of non-local map tasks is 23% and 17% of the total launched map tasks when executing jobs with 40 and 160 map tasks, respectively, and only 70% and 65% are successfully executed. Moreover, the percentage of the speculative tasks caused by non-local map tasks executions is 50% and 55% for the 40-maps and 160-maps jobs, respectively. Even worse, only 50% and 25% of the speculative map tasks are successful as shown in Figure 1 (c). We also observe that the response time of the job is almost proportional to the number of the non-local map tasks. In addition, the time gap between the non-local and local map tasks will affect the decision when speculation is needed. In our experiments, the non-local map tasks cause speculative executions with a probability of (2/8 = 25%) and (11/20 = 55%) for the 40-maps and 160-maps jobs, respectively. In contrast, the speculation caused by local map tasks is of a much lower probability, i.e., (2/32 = 6.5%) and (9/140 = 6.5%) for the same two jobs.

Furthermore, due to the time gap between the non-local and local map tasks along with the wrong decision when speculation, some nodes will be busy executing long-time map tasks ("slots occupying"), resulting with significant imbalance of the successful map tasks among different identical nodes. Figure 2 (a) shows the successful map tasks distribution among different data nodes when running *sort* benchmark with 160 map tasks. We can see that the variation is significant, the number of map tasks varies from 2 map tasks to 15 map tasks, which results in load imbalance in map tasks executions as shown in Figure 2 (b). Moreover, this situation induces load imbalance in the shuffle phase of the reduce phase, as some nodes need to transfer more intermediate data than others and will result in unbalanced load in reduce tasks executions as shown in Figure 2 (c).

III. Maestro Design

The goal of Maestro is to practically improve the performance of MapReduce (1) by reducing the number of non-local map tasks executions, and (2) by balancing the number of map tasks across different nodes. Getting the optimal solution for achieving the goal is difficult (an NP-hard problem). Therefore, we investigate for heuristic-based solutions. In particular, we consider the following heuristics:

- We always select the data node which we think it will execute the minimal number of local map tasks. We explain how to find these potential data nodes by calculating the probabilities of executing all the hosted chunks locally.
- We always select the data node which has minimal impact on other nodes' local map tasks executions. This node has the lowest *share rate* with other nodes. *Share rate* of a node n_j (denoted as $ShareRateN_j$) is defined as the maximum value of the number of replicas in n_j and n_i representing the same data chunks, for all n_i not equal to n_j in the cluster. $ShareRateN_j = max_{1 \leq i \leq N, i \neq j} Sc_i^j$, where Sc_i^j is the number of shared chunks between n_i and n_j, ($Sc_i^j = n_i \cap n_j$).
- We always select the chunk which has maximal probability of not being processed locally.

To facilitate the development of Maestro, we define the following two important parameters. We consider that the cluster consists of k servers: n_1, n_2, ..., n_k.

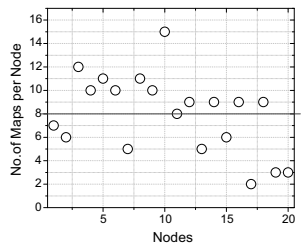
1) Number of hosted chunks in node n_j (denoted as HCN_j). The HCN value indicates the number of unprocessed chunks hosted by a node. The HCN values help in keeping the track of the chunks that are still unprocessed as a map task. After a task map_i is launched, the corresponding HCN of all the nodes hosting this specific chunk c_i and its replica are decreased by 1.

2) The chunk weight (denoted as Cw_i). The Cw value indicates the probability of processing c_i on none of its hosts (non-locally). Thus processing the chunk with the highest weight will reduce the risk of launched non local map tasks.

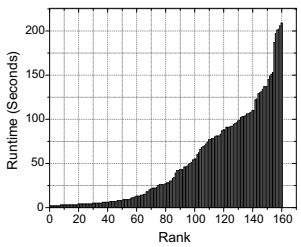
$$Cw_i = 1 - (\frac{1}{HCN_1} + \frac{1}{HCN_2} + ... + \frac{1}{HCN_r}) \quad (1)$$

where r donates the number of replications and HCN_j is the number of the chunks hosted by the data nodes which host c_i.

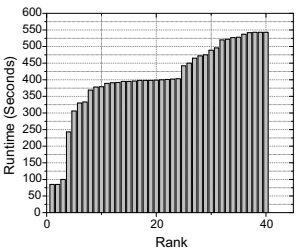
Fault Tolerance. Maestro is effective in handling failures, with the consideration of both the locality execution of the task and the priority of the map task executions. We calculate the Cw for the specific map task excluding the replica on



(a) Distribution of successful map tasks: *the number of successful map tasks per node varies by nearly 41%.*



(b) Map tasks runtime: *even though all map tasks receive the same amount of data, the slowest task takes more than 217 seconds while the fastest one completes in 2 seconds due to the non-local execution along with the imbalance of map task execution per node.*



(c) Reduce tasks runtime: *there is a factor of seven difference in runtime between the fastest and the slowest reduce tasks which is due to the map task-skew.*

Figure 2. Distribution of successful map tasks per nodes and tasks runtime: *Running sort benchmark with 160 map tasks and 40 reduces.*

the failed node, thus this map will be the first to be executed on any of the other hosts. As explained:

$$[Cw_i' = 1 - \sum_{j=1}^{k-1} \frac{1}{HCN_j}] > [Cw_i = 1 - \sum_{j=1}^{k} \frac{1}{HCN_j}] \quad (2)$$

where Cw_i' denotes the chunk weight when one of its replicas hosted by failed map task, hence sum of $(k-1)$ replicas, and Cw_i is the chunk weight with k replicas – no node failure.

Heterogeneous Cloud. In order to make Maestro effective for both homogeneous and heterogeneous environments[1], we extend the equation of calculating the chunk weight to:

$$Cw_i = 1 - (\frac{s_1}{HCN_1} + \frac{s_2}{HCN_2} + ... + \frac{s_r}{HCN_r}) \quad (3)$$

where s_j denotes the slots capacity of node n_j which reflects the heterogeneity degree. Accordingly, we select the chunk shared with nodes with smaller slots capacity (i.e. from equation (3), obviously, higher s_j leads to smaller Cw). We therefore prevent the nodes with higher computation capacity to be out of chunks "possible non-local maps" earlier than ones with lower computation capacity, especially that all the nodes are by default host the same number of chunks.

A. Maestro Scheduler

Considering the chunks locations and their replicas, Maestro schedules the map tasks considering chunk locality and node availability. The scheduling of Maestro is in two waves: first wave scheduler and run time scheduler.

1) First wave scheduler: In the current Hadoop's map scheduler, when the job is starting, all the nodes are treated equally and the map tasks will be scheduled blindly on these nodes with no specific order or condition which may cause early empty nodes (some nodes will be out of chunks). Therefore, in Maestro, we first process the nodes with higher potential in processing less local map tasks. For example, this may be the case when the number of hosted chunks is relatively smaller than on average or the nodes have low share rates. Accordingly, we process the nodes ascendingly by the sum of their hosted chunks' weights, while prioritizing nodes which share chunks with more nodes, (i.e., we process the nodes ascendingly according to their $NodeW_i$ values, where $NodeW_i = \sum_{j=1}^{HCN_i}(1 - \sum_{k=1}^{r} \frac{1}{HCN_k + Sc_k^i})$ where r is the number of replication. After selecting the node with minimal $NodeW_i$ value, we process the chunk with maximal weight. After this stage all the data nodes will be kept hosting nearly the same number of chunks.

2) Run Time Scheduler: When a node reports an empty task slot, the master will check if there is any unlaunched map task where the data is hosted by this specific node. Maestro then computes the Cw of each chunk and processes the chunk with the highest weight. We need to perform some refinements on the run time scheduler. We find in some applications, especially when the map task computation is fast (for example in *sort* benchmark where the map tasks execution time could be 3 seconds which is very close to the heartbeat time for some nodes) and when the slots number larger than 1, that some nodes may report an empty slot although they have reached the optimal number of local map tasks ($\frac{total\ Maps}{No.\ nodes}$), therefore instead of launching a non-local map task we firstly check for local speculative map tasks and then non-local map tasks.

In summary, the first wave scheduler is responsible for filling the empty slots of each data node based on the number of hosted map tasks and on the replication scheme for their input data, and the run-time scheduler is for runtime dynamics. They are complementary with each other, achieving the goals mentioned at the beginning of this section.

IV. Evaluations on Maestro

A. Experiments Setup

Maestro can be applied to Hadoop at different versions. Maestro is currently built in Hadoop-0.19.0 and Hadoop-0.21.0. We evaluate Maestro performance in two environments: on a local virtualized testbed – described in section

[1]In this paper, heterogeneity refers to heterogonous data nodes in terms of CPU cycles (number of CPUs pinned to one virtual machine).

Algorithm 1: First Wave Scheduler

Input: *D*: the needed data to be processed, *N*: number of data nodes, *S*: total available slots and *MapN*: all the map tasks.
Description: launch map tasks on *N* node to fill all the *S* free slots for the single MapReduce job.
Output: $\{map_i^{NL}(c_i, n_j)$, $map_i^{FL}(c_i, n_j)$, $map_i^{SL}(c_i, n_j)$,$map_i^{NN}(c_i, n_j)$, $map_i^{FN}(c_i, n_j)$, $map_i^{SN}(c_i, n_j)\}$ /*map^{XY} represents the type of map tasks where $X = \{N : Normal, F : Failed$, $S : Speculative\}$ and $Y = \{L : Local$, $N : Non - local\}$*/

```
while S is not zero do
    Find max_{1≤k≤N} s_k
    sort nodes according to their NodeW_j
    while s_j < max_{1≤k≤N} s_k do
        select next node
    end
    Find max_{1≤i≤HCN_j} Cw_i
    Launch map_i^{NL}(c_i, n_j)
    s_j ← s_j − 1
    foreach n_j host c_i do
        remove c_i
    end
    foreach n_j do
        calculate NodeW_j
    end
end
```

2 – and on a large scale cluster of Grid5000 [13]. Our Grid5000 experiments run on 100 nodes, located in the same site. Nodes are IBM eServer 326m with 2 cores 2.0GHz AMD Opteron 246, and 2GB of memory, equipped with 80GB/SATA disk. Table I summarizes the environments we use throughout our evaluation.

We configure the Hadoop-HDFS to host two replicas of each chunk. Each Hadoop-slave is configured to run 2 mappers and 2 reducers simultaneously (the default Hadoop). Throughout our evaluation, we use primarily the *sort* benchmark as testing workload (*sort* benchmark is the main benchmark used to evaluating MapReduce (Hadoop) [2, 7]), but also evaluate *wordcount* benchmark.

In all tests, data distribution is 128MB and 1GB per data node, equivalent to data set of 2.5GB and 10GB distributed on our local cluster, respectively, and 12.5GB and 200GB

Algorithm 2: Run-Time Scheduler

```
When a heartbeat is received from node n_j reporting free slot
if MapN_j is not empty then
    Find max_{1≤i≤HCN_j} Cw_i
    Launch map_i^{NL}(c_i, n_j)
else if n_j host speculative map task input then
    Launch map_i^{SL}(c_i, n_j)
else
    Launch map_i^{SN}(c_i, n_j)
end
```

Table I
ENVIRONMENTS USED IN OUR EVALUATION

Environments	Scale
Local Cluster	4VMs/Node equivalent to 20VMs cluster
Grid5000	100-nodes

on Grid5000.

B. Maestro Scheduling on Local Cluster

We first evaluate Maestro and native Hadoop on our local virtual cluster of 20VMs. We compare the response time – the average of three runs of the job – and the balance of the successful map tasks among data nodes in Maestro against native Hadoop. We use two metrics to measure the balance of map tasks distribution [16]:

- the coefficient of variation:

$$cv = \frac{stdev}{mean} \times 100\% \quad (4)$$

- The min-max ratio:

$$Min - Max\ Ratio = \frac{\min map_i}{\max map_j} \times 100\% \quad (5)$$

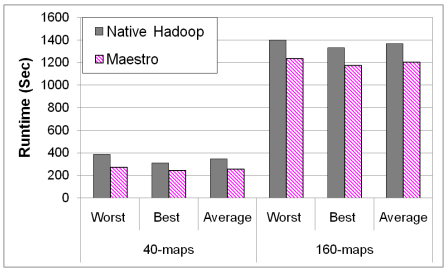
where $\min map_i$ and $\max map_j$ are the minimum and the maximum number of the successful map tasks among different nodes.

We use two *sort* jobs on two data sets of 2.5GB and 10GB, equivalent to 40 and 160 map tasks, respectively. Figure 3 (a) shows the worst, best and average runtime achieved by Maestro in contrast to native Hadoop. On average, Maestro outperforms Hadoop by 25% and 11.8% for 40-maps and 160-maps, respectively. This can be explained due to the decreasing number of non-local map tasks, (e.g. the number of non-local map tasks is 23% in Hadoop, and it is only 4% in Maestro for the *sort* job with 160-maps). Moreover, the speculative maps ratio ($speculative\ maps\ ratio = \frac{speculative\ map\ tasks}{total\ successful\ map\ tasks}$) is significantly decreased from 10% to nearly 1.25% for 40-maps and from 12.5% to 2.5% for 160-maps.

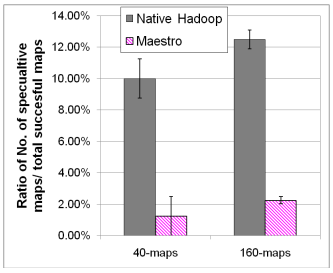
Table II shows the variation of the successful map tasks executions on different nodes. We can see that the variation is significant in native Hadoop compared to Maestro. For example, for 160-maps, the co-efficient of variation is almost 42% and the min-max ratio is 14.2% in native Hadoop while there are 16.2% and 55.33% in Maestro, respectively.

C. Maestro Scheduling on Grid5000

In order to validate our evaluation on large scale distributed system, we evaluate Maestro and native Hadoop on large scale cluster of 100-nodes on Grid5000. We compare Maestro and native Hadoop in two settings: homogeneous and heterogeneous environments. In order to mimic heterogeneous environment, we divide the nodes into two sets: one with 2 map task slots capacity and one with 1 map task slot capacity.



(a) Response time of *sort* on 20-VMs local cluster: *worst, best and average case*



(b) Speculative maps ratio

Figure 3. *Sort* benchmark with Maestro against native Hadoop on our local cluster

Table II
VARIATION OF SUCCESSFUL MAP TASKS AMONGST DIFFERENT NODES FOR MAESTRO AGAINST NATIVE HADOOP

		cv		Min-Max Ratio	
		Hadoop	*Maestro*	*Hadoop*	*Maestro*
Local cluster	*40-maps*	39.7%	16.2%	25%	33.3%
	160-maps	41.7%	16.2%	14.2%	55.3%
Grid5000	*200-maps*	18%	12%	33%	33%
	1600-maps	25%	11%	43%	72%

As shown in Figure 4, Maestro outperforms Hadoop by 34% and 6% for 200-maps and 1600-maps, equivalent to 128MB, and 1GB data distribution, respectively. The number of non-local map tasks is 16% in Hadoop, and it is only 3% in Maestro. Moreover, for the 1600-maps, the speculative map tasks is significantly decreased from 5% to 1% of the total map tasks. Moreover, Maestro leads to a better balanced distribution of successful map tasks among different nodes. As shown in Table II, the maestro reduced the *cv* from 18% to 12% for 200-maps and from 25% to 11% for 1600-maps. Furthermore, the min-max ratio is significantly improved – Maestro reduces the gap between the node with the maximum number of successful map tasks and the one with the minimum number to 72%, while it is 43% in native Hadoop for 1600-maps. In addition, the reduction in both non-local map tasks and speculative map tasks along with improving the balance of successful map tasks result with shorter and more predictable map task phase in contrast with native Hadoop and relatively shorter reduce phase as shown in Figure 4.

1) Maestro scheduling with the wordcount application: We run *wordcount* application to evaluate Maestro. As shown in Figure 5, on average, Maestro allows the *wordcount* application to run 22% and 7% faster than Hadoop for 200-maps and 1600-maps, respectively. Moreover, the non-local map tasks reduced by 77% from 11% in native Hadoop to 2.4% in Maestro. The speculation reduced from 5.3% in native Hadoop to only 1.1% in Maestro. The improvement in *sort* benchmark is relatively better than *wordcount* when the number of map tasks is small, because the local map task execution time is much shorter compared to non-local map tasks in *sort* application compared to *wordcount* application.

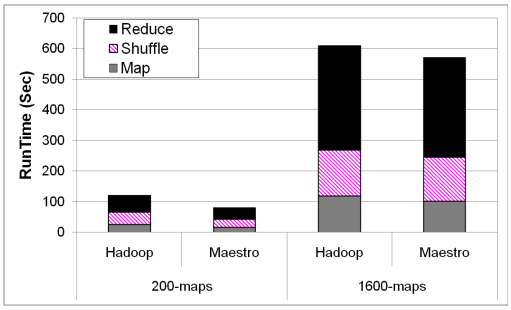


Figure 4. *Sort* on Grid5000: *detailed performance of each stage in Maestro against native Hadoop*

2) Maestro scheduling in heterogeneous cluster: Similar to the homogeneous environments, Maestro can be efficiently employed in heterogeneous environment to improve the local execution of map tasks and reduce the needless map tasks speculation, as a result, improve the overall performance of the applications. As shown in Figure 6, Maestro outperforms Hadoop by 24% and 4% for 200-maps and 1600-maps when running *sort* application, respectively. The number of non-local map tasks is 14% in Hadoop, while it is only 4% in Maestro. Moreover, for the 1600-maps, the number of speculative map tasks is significantly decreased by 70% – from 88 speculative map tasks in native Hadoop to 27 speculative map tasks in Maestro. In addition, Maestro allows the *wordcount* application to run 13% and 6% faster than Hadoop for for 200-maps and 1600-maps, respectively, and the speculation reduced from 70 speculative map tasks in native Hadoop to 26 speculative map tasks in Maestro for the *wordcount* application with 1600-maps.

V. RELATED WORK

Ever since the advent of the MapReduce programming model, a huge number of studies have been dedicated to improving the performance of MapReduce system, and Hadoop in particular [7, 8, 17, 18]. A series of network-oriented optimizations focus on the data shuffling stage of MapReduce [19–21]. Tyson *et al.* [19] have proposed

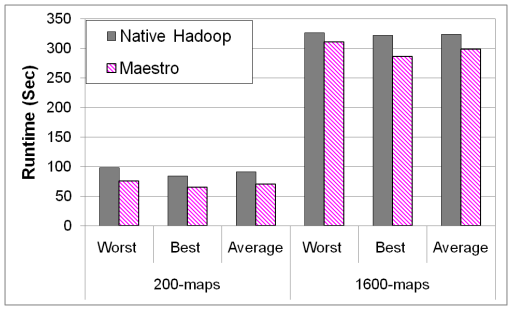


Figure 5. *Wordcount* on Grid5000: *worst, best and average-case time for Maestro against native Hadoop*

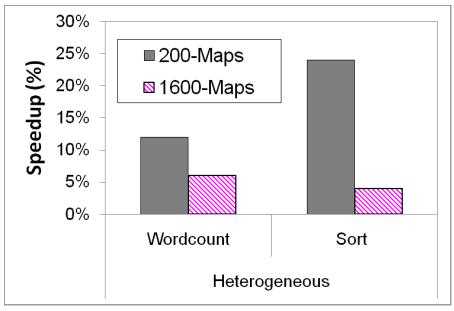


Figure 6. Maestro speed up against native Hadoop on heterogeneous Grid5000: *running two benchmarks, sort and worcount, on heterogeneous platform on Grid5000 with two data sets*

Online MapReduce, a modified MapReduce architecture in which intermediate data are pipelined between map and reduce tasks, so that reducers start processing data as soon as mappers produce them. This in turn results with faster runtime. Sangwon *et al.* [20] have proposed pre-fetching and pre-shuffling schemes for MapReduce in shared environment. The pre-shuffling scheme tries to reduce the network overhead required to shuffle intermediate data, by looking over the input splits before the map phase begins and predicts the target reducer where the key-value pairs of the intermediate output are partitioned; accordingly, the expected data are assigned to a map task near the future reducer before the execution of the mapper. We have proposed LEEN [21], a solution to reduce the network overhead required to shuffle intermediate data, by partitioning the intermediate data considering data locality in the mapper outputs while keeping in mind fair distribution of reducers' inputs.

There have been a few studies on improving MapReduce performance by overcoming the performance degradation caused by virtualization interference [6, 7, 17] (such interferences are contributed by different factors including the application's type, the number of concurrent VMs, and the scheduling algorithms used within both the guest and the host [22]). We have proposed CLOULET [6], a new framework for the MapReduce model by decoupling the storage unit, namely HDFS, from the computation unit, namely VMs, and keeping the data transfer physical node based, thus minimizing the impact of the I/O virtualization on the Hadoop execution. Zaharia *et al.* [7] have proposed a new scheduling algorithm called *Longest Approximate Time to End* (LATE) to improve the performance of Hadoop in a heterogeneous environment, brought by the variation of VM consolidation amongst different physical machines, by preventing the incorrect execution of "speculative tasks". Although, the focus of LATE is to prevent the incorrect speculative reduce tasks for short jobs in heterogeneous cloud, Maestro aims to achieve the same aim but for speculative map tasks regardless the targeting environment.

Recent studies have demonstrated the performance gain of Hadoop through data placement techniques [23, 24]. Xie *et al.* [23] have proposed to distribute the input data chunks to heterogeneous nodes based on their computing capacities. Ananthanarayanan *et al.* [24] have proposed Scarlett, a popularity-based data replication technique to reduce slots contention on machines storing popular data and maximize data locality. The monetary cost of running Hadoop has been explored on public cloud environments [25, 26]. Cohan *et al.* [26] has proposed to improve the performance of Hadoop in public cloud with minimal monetary cost increasing by adding spot instances as accelerators. We plan to investigate the monetary efficiency of Maestro in the future.

A closely related work on improving the local execution of map tasks in Hadoop is delay scheduler [27]. In [27], Zaharia *et al.* have proposed a simple scheduling algorithm called delay scheduling to achieve locality and fairness in cluster scheduling. When a job that should be scheduled next according to fairness cannot launch a local task, it waits for small amount of time, letting other jobs launch tasks instead. Our work is different in the targeting environment, we focus on batch jobs with the awareness of replications. Moreover, delay scheduler may increase the locality of map tasks executions for single job but at the cost of performance and number of speculative map tasks. Because it will introduce a delay to enforce locality, this will lead to increase the overall response time and will cause incorrect map speculation, especially for applications with short map task execution.

VI. Summary and Future Work

As data-intensive applications became popular in the cloud, data-intensive cloud systems call for empirical evaluations and technical innovations. In this study, we investigate some performance limits in current MapReduce frameworks (Hadoop in particular). Our studies reveal that the current Hadoop's scheduler for map tasks is inadequate, as it disregards replicas distributions. It causes performance degradation due to a high number of non-local map tasks, which in turn causes too many needless speculative map tasks and leads to imbalanced execution of map tasks among data nodes. We address these problems through developing a new

map task scheduler called Maestro. Maestro is conducive to improving the locality execution of map tasks efficiency by the virtue of the fine-grained replica-aware execution of map tasks. Our preliminary results with Maestro demonstrate promising improvements compared to Hadoop. Our future work lies in two aspects: first, to improve the Maestro algorithm to work in dynamic settings in public cloud such as Amazon EC2; second, to evaluate our implementation in shared environments with the possible integration with existing schedulers such as the fair scheduler [27].

ACKNOWLEDGMENT

This work is supported by NSF of China under grant No.61133008, the Outstanding Youth Foundation of Hubei Province under Grant No.2011CDA086, the International Cooperation Program of Wuhan Technology Bureau under grant 201171034311, the National Science & Technology Pillar Program of China under grant No.2012BAH14F02, the Inter-disciplinary Strategic Competitive Fund of Nanyang Technological University 2011 No.M58020029, and the ANR MapReduce grant (ANR-10-SEGI-001). This work was done in the context of the Héméra INRIA Large Wingspan-Project (see http://www.grid5000.fr/mediawiki/index.php/Hemera).

The experiments presented in this paper were carried out using the Grid5000/ALADDIN-G5K experimental testbed, an initiative of the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see http://www.grid5000.fr/).

REFERENCES

[1] 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 ACM European Conference on Computer Systems (EuroSys '07)*, Lisbon, Portugal, 2007, pp. 59–72.

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th USENIX conference on Symposium on Opearting Systems Design & Implementation (OSDI '04)*, San Francisco, CA, USA, 2004, pp. 137–150.

[3] The Apache Hadoop Project, http://www.hadoop.org.

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

[5] D. Gottfrid, "Self-service, Prorated Super Computing Fun!" http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/.

[6] S. Ibrahim, H. Jin, B. Cheng, H. Cao, S. Wu, and L. Qi, "Cloudlet: towards mapreduce implementation on virtual machines," in *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC-18)*, Garching, Germany, 2009, pp. 65–66.

[7] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI'08)*, San Diego, California, 2008, pp. 29–42.

[8] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, Vancouver, Canada, 2008, pp. 1099–1110.

[9] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou, "Comet: batched stream processing for data intensive distributed computing," in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*, Indianapolis, Indiana, USA, 2010, pp. 63–74.

[10] W. Fang, B. He, Q. Luo, and N. K. Govindaraju, "Mars: Accelerating mapreduce with graphics processors," *IEEE Trans. Parallel Distrib. Syst.*, Vol. 22, pp. 608–620, 2011.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SIGOPS - Operating Systems Review*, Vol. 37, No. 5, pp. 29–43, 2003.

[12] Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Iréa, "Grid'5000: a large scale and highly reconfigurable experimental Grid testbed." *International Journal of High Performance Computing Applications*, Vol. 20, No. 4, pp. 481–494, 2006.

[13] Grid 5000 Project, https://www.grid5000.fr/mediawiki/index.php.

[14] Xensource Homepage,, http://www.xensource.com/.

[15] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, 2003, pp. 164–177.

[16] R. Jain, D.-M. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *DEC Research Report TR-301*, 1984.

[17] S. Ibrahim, H. Jin, L. Lu, B. He, and S. Wu, "Adaptive disk i/o scheduling for mapreduce in virtualized environment," in *Proceedings of the 2011 International Conference on Parallel Processing (ICPP'11)*, Taipei, Taiwan, 2011, pp. 335–344.

[18] B. He, M. Yang, Z. Guo, R. Chen, W. Lin, B. Su, H. Wang, and L. Zhou, "Wave computing in the cloud," in *Proceedings of the 12th USENIX Workshop on Hot Topics in Operating Systems (HotOS'09)*, Monte Verità, Switzerland, 2009.

[19] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*, San Jose, California, 2010.

[20] S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, and S. Maeng, "Hpmr: Prefetching and pre-shuffling in shared mapreduce computation environment," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing (CLUSTER'07)*, New Orleans, Louisiana, USA.

[21] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CLOUDCOM'10)*, Indianapolis, USA, 2010, pp. 17–24.

[22] S. Ibrahim, B. He, and H. Jin, "Towards pay-as-you-consume cloud computing," in *Proceedings of the 2011 IEEE International Conference on Services Computing (SCC'11)*, Washington, DC, USA, 2011, pp. 370–377.

[23] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *Proceedings of the 24th IEEE International Symposium on Parallel and Distributed Workshop (IPDPSw'10)*, Atlanta, Georgia, USA, 2010, pp. 1–9.

[24] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," in *Proceedings of the sixth ACM European Conference on Computer Systems (EuroSys'11)*, Salzburg, Austria, 2011, pp. 287–300.

[25] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10)*, Boston, MA, 2010.

[26] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, Boston, MA, 2010.

[27] 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 ACM European Conference on Computer Systems (EuroSys'10)*, Paris, France, 2010, pp. 265–278.