# On Optimal Budget-Driven Scheduling Algorithms for MapReduce Jobs in the Heterogeneous Cloud

Yang Wang [♭] and Wei Shi [♯]

[♭]*IBM Center for Advanced Studies (CAS Atlantic)*
*University of New Brunswick, Fredericton, Canada E3B 5A3*
[♯] *Faculty of Business and Information Technology*
*University of Ontario Institute of Technology, Ontario, Canada*
*E-mail: yangwang@ca.ibm.com, wei.shi@uoit.ca*

***Abstract*—In this paper, we consider task-level scheduling algorithms with res-pect to budget and deadline constraints for a bag of MapReduce jobs on a set of provisioned heterogeneous (virtual) machines in cloud platforms. Heterogeneity is manifested in the ”pay-as-you-go” charging model we use, where service machines with different performance have different service rates. We organize the bag of jobs as a $\kappa$-stage workflow and achieve, for specific optimization goals, the following results. First, given a total monetary budget $B_j$ for a particular stage $j$, we propose a greedy algorithm for distributing the budget, with minimal stage execution time as our goal. Based on the structure of this problem, we further prove the optimality of our algorithm in terms of the budget used and the execution time achieved. We then combine this algorithm with dynamic programming techniques to propose an optimal scheduling algorithm that obtains a minimum scheduling length in $O(\kappa B^2)$. The algorithm is efficient if the total budget $B$ is polynomially bounded by the number of tasks in the MapReduce jobs, which is usually the case in practice. Second, we consider the dual of this optimization problem to minimize the cost when the (time) deadline of the computation $D$ is fixed. We convert this problem into the standard multiple-choice knapsack problem via a parallel transformation. Our empirical studies verify the proposed optimal algorithms.**

***Keywords*-Heterogeneous Clouds, MapReduce optimization, optimal Hadoop scheduling algorithm, budget constraints**

## I. Introduction

Clouds, with its abundant on-demand compute resources and elastic charging models, have emerged as a promising platform to address various data processing and task computing problems [1]–[3]. On the other hand, MapReduce [4], characterized by its magnificent simplicity, fault tolerance, and scalability, is becoming a popular programming framework in Cloud to automatically paralellize large scale data processing as web indexing, data mining [5], and bioinformatics [6]. Since the cloud supports on-demand “massively parallel” applications with loosely coupled computational tasks, it is amenable to the MapReduce framework and thus suitable for diverse MapReduce applications. Therefore, many cloud infrastructure providers have deployed the MapReduce framework on their commercial clouds (e.g., Amazon Elastic MapReduce (Amazon EMR)) as one form of infrastructure services.

Because MapReduce (MR for brevity) offers a fast and powerful solution in many distinct areas, it is of interest to cloud service providers (CSPs). That is, *MR as a Service* (MRaaS) is software that can typically be set up on a provisioned MR cluster on cloud instances. However, to reap the benefits of such a service, many challenging problems have to be addressed. Most current studies focus solely on system issues in deployment, such as overcoming the limitations of the cloud infrastructure to build up the framework [7], [8], evaluating the performance loss running the framework on virtual machines [9], and other issues in fault tolerance [10], reliability [11], data locality [12] and so on. We are also aware of some recent research addressing the scheduling problem of MR in Clouds [13]–[17]. However, these contributions mainly addressed the scheduling issues with respect to various concerns in dynamic loading [14], energy reduction [16], and network performance. [17]. No one has yet optimized the scheduling of MR jobs at task level with respect to budget constraints. We believe this situation proceeds from several factors. First, as discussed above, the MR service is charged together with other infrastructure services. Second, due to some properties of the MR framework (such as automatic fault tolerance with speculative execution [18]), it is sometimes hard for CSPs to account a job execution in a reasonable way.

Since cloud resources are typically provisioned on demand with a billing model of ”pay-as-you-go”, cloud-based applications are usually budget driven. Consequently, when deploying the MR framework as a MRaaS, the efficient use of the resources to fulfil the performance requirements within budget constraints is always a concern for the CSPs in practice. S. Ibrahim et al. [9] evaluated the performance degradation of MapReduce on virtual machines (VMs). To address this problem, they also proposed a novel MapReduce framework on virutal machines called *Cloudlet* to overcome the overhead of VM while benefiting from the other features of VM (e.g., management and reliability issues). In this paper, we investigate the problem of scheduling a bag of MR jobs with budget and deadline constraints in heterogeneous

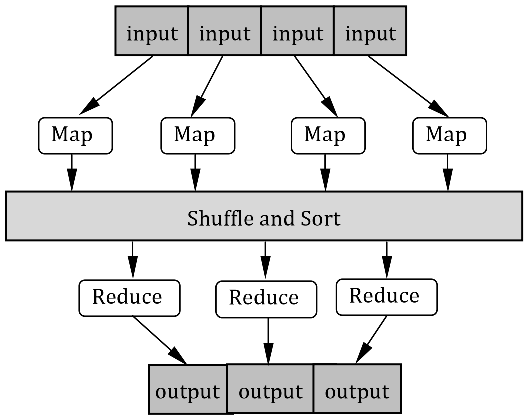


Figure 1: MapReduce framework.

clouds. The bag of MR jobs could be an iterative MR job, a set of independent MR jobs, or a collection of jobs related to some high-level applications such as Hadoop Hive [19].

A MR job at a high level essentially consists of two sets of tasks, map tasks and reduce tasks as shown in Figure 1. The executions of both sets of tasks are synchronized into different stages, sequentially. In the map stage, the entire dataset is partitioned into several smaller chunks in the form of key-value pairs, each chunk being assigned to a map node for partial computation results. The map stage ends up with a set of intermediate key-value pairs on each map node, which are further shuffled based on the intermediate keys into a set of scheduled reduce nodes where the received pairs are aggregated to obtain the final results. For an iterative MR job, the final results could be tentative and further partitioned into a new set of map nodes for the next round of the MR computation.

A bag of MR jobs may have multiple stages of MR computation, each stage running either map or reduce tasks in parallel, with enforced synchronization only between them. Therefore, the executions of the jobs can be viewed as a fork&join workflow characterized by multiple synchronized stages, each consisting of a collection of sequential or parallel map/reduce tasks. An example of such a workflow is shown in Figure 2, which is composed of 4 stages, each with 8, 2, 4 and 1 (map/reduce) tasks. These tasks are scheduled on different nodes for parallel execution. In heterogeneous clouds, each node, depending on the performance or the configuration, could have different service rates. As the resources of the Cloud computing are on-demand provisioned according to the typical ”pay-as-you-go” model, the cost-effective selection and utilization of the resources for each running task are thus a pragmatic concern of the CSPs to compute their MR workloads, especially the computation budget is fixed.

In this paper, we are interested in the cost-effective scheduling of such a bag of MR jobs on a set of heterogeneous (virtual) machines in Clouds, each with different

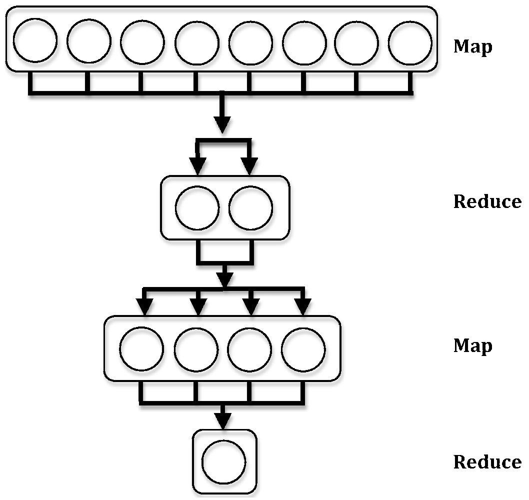


Figure 2: An example of a 4-stage iterative MapReduce job.

performance and service rates, in an efficient way. We call this scheduling *task-level scheduling*, which is fine grained compared to the frequently discussed job-level scheduling, where the scheduled unit is a job instead of a task. More specifically, we concern ourselves with the following two optimization problems:

1) Given a fixed budget $B$, how to efficiently select the machine from a candidate set for each task so that the total scheduling length of the job is shortest without breaking the budget;
2) Given a fixed deadline $D$, how to efficiently select the machine from a candidate set for each task so that the total monetary cost of the job is lowest without missing the deadline;

At the first sight, both problems are primary-dual each other; solving one is sufficient to solve the other. However, we will show that there are still some asymmetries in their solutions. In particular, we focus squarely on the first problem, and briefly discuss the second.

To address the first problem, we first design an efficient greedy algorithm for computing the minimum execution time with a given budget for each stage. Based on the structure of this problem and the adopted greedy choice, we further prove the optimality of this algorithm in terms of the execution time and consumed budget. With these results, we further develop a dynamic programming algorithm to achieve a global optimal solution within a time complexity of $O(\kappa B^2)$. In contrast, the second problem is relatively straightforward as we can change it into the standard *multiple-choice knapsack* (MCKS) problem [20] via a parallel transformation. Our results show that the problems can be efficiently solved if the total budget $B$ or the deadline $D$ are polynomially bounded by the number of tasks or the number of stages in the workflow, which is usually the case in practice.

The rest of this paper is organized as follows: in Section II, we introduce some background knowledge regarding to the MR framework and survey some related work. Section III is our problem formulation. The proposed budget-driven algorithms are presented in Section IV and their empirical studies are in the following Section V. Finally, we conclude the paper in Section VI.

## II. Background and Related Work

The MR framework was first advocated by Google in 2004 as a programming model for its internal massive data processing [21]. Since then it has been widely discussed and soon accepted as the most popular paradigm for data intensive processing in different contexts. There are many implementations of this framework in both industry and academia such as Hadoop [22], Dryad [23], Greenplum [24].

As the most popular open source implementation, Hadoop MR has become the de facto research prototype on which many related studies are conducted. Therefore, we use the terminology of the Hadoop community in the rest of this paper, and in particular, most of the related work we overview in this section are built up on the Hadoop implementation.

Hadoop MR is made up of an execution runtime and a distributed file system. The execution runtime is responsible for job scheduling and execution, which is composed of one master node called *JobTracker* and some slave nodes called *TaskTrackers*. In contrast, the distributed file system, called *HDFS*, is used to manage task and data across nodes. When the JobTracker receives a submitted job, it first splits the job into a number of map and reduce tasks and then allocates them to the TaskTrackers. As with most distributed systems, the performance of the task scheduler has great impact on the scheduling length of a job, in our particular case, also the budget consumed.

Hadoop MR provides a FIFO-based default scheduler at job level while at task level, it provides a TaskScheuler interface, which allows developers to design their own schedulers. By default, each job would occupy the whole cluster and execute in order. To overcome this defect and fairly share the cluster among jobs and users over time, Facebook and Yahoo! leveraged the interface to implement Fair Scheduler [25] and Capacity Scheduler [26], respectively.

In addition to fairness, research on the scheduler of Hadoop MR with respect to different opportunities for improving the scheduling policies exists. For instance, Hadoop adopts *speculative task scheduling* to minimize the slowdown in the synchronization phases caused by straggling tasks in a homogeneous environment [22]. To extend this idea to heterogeneous clusters, Zaharia et al. [18] proposed the LATE algorithm. But their algorithm does not consider the phenomenon of dynamic loading, which is common in practice. This issue was studied by You et al. [14] and they proposed a load-aware scheduler. Another instance is the *delay scheduling* mechanism which was developed in [12] to improve data locality with relaxed fairness. Apart from these, other research efforts include *power-aware scheduling* [27], *deadline constraint scheduling* [28], and scheduling based on automatic task slot assignments [29] to quote a few. Although these results improve different aspects of MR scheduling, they are mostly system performance-centric. It appears that no one yet shares our goal of focusing on the monetary cost of the problem.

However, there are some similar studies in the context of scientific workflow scheduling on HPC platforms including the Grid and Cloud [30]–[32]. Yu et al. [30] discussed this problem based on service Grids where a QoS-based workflow scheduling method was presented to minimize the execution cost and yet meet the time constraints imposed by the user. In contrast, Zeng et al. [31] considered the executions of large scale many-task workflows in Clouds with budget constraints. To effectively balances the execution time-and-monetary costs, they proposed *ScaleStar*, a budget-conscious scheduling algorithm, which assigns the selected task to a virtual machine with higher comparative advantage.

Almost at the same time, Caron et al. [32] studied the same problem for non-deterministic workflows. They presented a way of transforming the initial problem into a set of addressed sub-problems thereby proposing two new allocation algorithms for resource allocations under budget constraints. All these studies focus on the scheduling of scientific workflows with a deterministic or non-deterministic DAG (directed acyclic graph) shape. In this sense, in our context, the abstracted fork&join workflow can be viewed as a special case of general workflows. However, our focus is on MR scheduling within budget and deadline constraints, rather than on the general workflow scheduling problem.

## III. Problem Formulation

We model an iterative MR job as a multi-stage fork&join workflow that consists of $\kappa$ stages (called $\kappa$-stage job), each stage $j$ having a collection of independent map or reduce tasks, denoted as $J_j = \{J_{j0}, J_{j1}, ..., J_{jn_j}\}$, $0 \le j < \kappa$, and $n_j + 1$ is the size of stage $j$. In a cloud platform, each map or reduce task may be associated with a set of machines that are provided by the cloud infrastructure provider to run that task, each depending on the performance, with different charge rates. More specifically, for Task $J_{jl}$, $0 \le j < \kappa$ and $0 \le l \le n_j$ the available machines and corresponding prices (service rates) are listed in Table I.

$$\begin{bmatrix} t_{jl}^1 & t_{jl}^2 & \dots & t_{jl}^{m_{jl}} \\ p_{jl}^1 & p_{jl}^2 & \dots & p_{jl}^{m_{jl}} \end{bmatrix}$$

Table I: Time-price table of task $J_{jl}$

Table II: Notation frequently used in model and algorithm descriptions

| *Symbol* | *Meaning* |
|---|---|
| $\kappa$ | the number of stages |
| $J_{ji}$ | the ith task in stage $j$ |
| $J_j$ | task set in stage $j$ |
| $n_j$ | the number of tasks in stage $j$ |
| $n$ | the total number of tasks in the workflow |
| $t_{jl}^u$ | the time to run task $J_{jl}$ on machine $M_u$ |
| $p_{jl}^u$ | the cost rate for using $M_u$ |
| $m_{jl}$ | the total number of the machines that can run $J_{jl}$ |
| $m$ | the total size of time-price tables of the workflow |
| $B_{jl}$ | the budget used by $J_{jl}$ |
| $B$ | the total budget for the MapReduce job |
| $T_{jl}(B_{jl})$ | the shortest time to finish $J_{jl}$ given $B_{jl}$ |
| $T_j(B_j)$ | the shortest time to finish stage $j$ given $B_j$ |
| $T(B)$ | the shortest time to finish the job given $B$ |
| $D_j$ | the deadline to stage $j$ |
| $C_{jl}(D_j)$ | the minimum cost to finish $J_{jl}$ in stage $j$ within $D_j$ |
| $C(D)$ | the minimum cost to finish the job within $D$ |

In this table, $t_{jl}^u, 1 \le u \le m_{jl}$ represents the time to run task $J_{jl}$ on machine $M_u$ whereas $p_{jl}^u$ represents the corresponding price for using that machine, and $m_{jl}$ is the total number of the machines that can run $J_{jl}$. Here, time is sorted in increasing order and the prices in decreasing order. Without loss generality, we assume that the values of time and prices are unique in their own sequence. This assumption is based on the observation that, when two machines have the same run time for a task, no one will select the expensive one. Similarly, for any two machines with same price, no one will select the slow machine to run a task. Some frequently used symbols are then listed in Table II for quick reference in the following description.

*1) Budget Constraints:* Given budget $B_{jl}$, the shortest time to finish task $J_{jl}$, denoted as $T_{jl}(B_{jl})$, is defined as

$$T_{jl}(B_{jl}) = t_{jl}^u \qquad p_{jl}^{u+1} < B_{jl} < p_{jl}^{u-1} \tag{1}$$

Obviously, if $B_{jl} < p_{jl}^{m_{jl}}$, $T_{jl}(B_{jl}) = +\infty$.

The time to complete a stage $j$ with budget $B_j$, denoted as $T_j(B_j)$, is defined as the time the last task in that stage is finished by using the given budget,

$$T_j(B_j) = \max_{\sum_{l\in[0,n_j]} B_{jl} \le B_j} \{T_{jl}(B_{jl})\} \tag{2}$$

For fork&join, one stage cannot start until its immediately preceding stage has finished. Thus the total makespan with budget $B$ to complete the workflow is defined as the sum of all stages' time. Our goal is to minimize the time within the given budget $B$.

$$T(B) = \min_{\sum_{j\in[0,\kappa)} B_j \le B} \sum_{j\in[0,\kappa)} T_j(B_j) \tag{3}$$

*2) Deadline Constraints:* Given deadline $D_j$ for stage $j$, the minimum cost to finish stage $j$ is

$$C_j(D_j) = \sum_{l\in[0,n_j]} C_{jl}(D_j) \tag{4}$$

where $C_{jl}(D_j)$ is the minimum cost to finish $J_{jl}$ in stage $j$ within $D_j$. Note that $t_{jl}^1 \le D_j \le t_{jl}^{m_{jl}}$, otherwise $C_{jl}(D_j) = +\infty$. Our optimization problem can be written as

$$C(D) = \min_{\sum_{j\in[1,k]} D_j \le D} \sum_{j\in[0,\kappa)} C_j(D_j) \tag{5}$$

Some readers may argue the feasibility of this model since the number of stages and the number of tasks in each stage needs to be known a prior. Actually, it is feasible in reality because the number of map tasks for a given job is driven by the number of input splits, which is known to the scheduler, while for the number of reduce tasks, it can be pre-set with parameters (e.g.,mapred.reduce.tasks in Hadoop). As for the number of stages, it is not always possible to pre-define it for iterative MR jobs. This is a limitation of our model. But for a set of independent jobs, we can treat them, under the default FIFO job scheduler, as a single fork&join workflow. Therefore, our model is still representative of most cases in reality.

## IV. Optimal Budget-Driven Scheduling Algorithm

In this section, we propose our task-level scheduling algorithms for iterative MR jobs with the goal of optimizing Equations (3) and (5) within budget and deadline constraints. We first consider the optimization problem under budget constraint, and leverage dynamic programming techniques to obtain an optimal solution. To overcome the inherent complexity of the optimal solution, we also present an efficient greedy algorithm. Given the results of the budget constraints, the variant of this problem when a deadline constraint is imposed to minimize the monetary cost is also discussed.

### *A. Optimization under Budget Constraints*

The proposed algorithm should have the capability of distributing the budget among the stages, and in each stage distributing the assigned budget to each constituent task in an optimal way. To this end, we design the algorithm in two steps:

1) Given budget $B_j$ for stage $j$, distribute the budget to all constituent tasks in such a way that $T_j(B_j)$ is minimum (i.e., Equation (2)). Clearly, each such computation is stage-wise and independent of other stages. Therefore these computations can be computed in parallel using $\kappa$ machines.
2) Given budget $B$ for the workflow and the results in Equation (2), optimize Equation (3).

*1) Per-Stage Distribution:* To address the problem in the first step, we develop a greedy algorithm to distribute budget $B_j$ between the $n_j + 1$ tasks in such a way that $T_j(B_j)$ is minimized. Based on the structure of this problem, we then prove the optimality of the greedy algorithm.

**Algorithm 1** per-stage distribution algorithm

```
procedure $T_j(n_j, B_j)$                                   ▷ Dist. $B_j$ among $J_j$
    $B'_j = B_j - \sum_{l\in[0,n_j]} p_{jl}^{m_{jl}}$
    if $B'_j < 0$ then return $(+\infty)$
    end if
                                                            ▷ Initialization
    for $J_{jl} \in J_j$ do                                 ▷ $O(n_j)$
        $T_{jl} \leftarrow t_{jl}^{m_{jl}}$                 ▷ record exec. time
        $B_{jl} \leftarrow p_{jl}^{m_{jl}}$                 ▷ record budget dist.
        $M_{jl} \leftarrow m_{jl}$                          ▷ record assigned machine.
    end for
    while $B'_j \geq 0$ do                                  ▷ $O(\frac{B_j \log n_j}{\min_{0\leq l\leq n_j}\{\delta_{jl}\}})$
        $jl^* \leftarrow \arg\max_{l\in[0,n_j]}\{T_{jl}\}$  ▷ get the slowest task
        $u \leftarrow M_{jl^*}$
        if $u = 1$ then
            return $(T_{jl^*})$
        end if
                                                            ▷ Lookup matrix in Table I
        $< p_{jl^*}^{u-1}, p_{jl^*}^{u} > \leftarrow$ Lookup$(J_{jl^*}, u-1, u)$
        $\delta_{jl^*} \leftarrow p_{jl^*}^{u-1} - p_{jl^*}^{u}$
        if $B'_j \geq \delta_{jl^*}$ then                   ▷ reduce $J_{jl^*}$'s time
            $B'_j \leftarrow B'_j - \delta_{jl^*}$
                                                            ▷ Update
            $B_{jl^*} \leftarrow B_{jl^*} + \delta_{jl^*}$
            $T_{jl^*} \leftarrow t_{jl}^{u-1}$
            $M_{jl^*} \leftarrow u - 1$
        else
            return $(T_{jl^*})$
        end if
    end while
end procedure
```

The idea of the algorithm is simple. To ensure that all the tasks in stage $j$ have sufficient budget to finish while minimizing $T_j(B_j)$, we first require $B'_j = B_j - \sum_{l\in[0,n_j]} p_{jl}^{m_{jl}} \geq 0$ and then iteratively distribute $B'_j$ in a greedy manner each time to the task whose current execution time determines $T_j(B_j)$ (i.e., the slowest one). This process continues until no sufficient budget is left over. Algorithm 1 shows the pseudo code of the algorithm.

In the algorithm, we use three *profile variables* $T_{jl}, B_{jl}$ and $M_{jl}$ for each task $J_{jl}$ in order to record its execution time, assigned budget, and the selected machine (Lines 6-10). After setting these variables with their initial values, the algorithm enters into its main loop to iteratively update the profile variables associated with the current slowest task (i.e.,$J_{jl^*}$) (Lines 11-30). By searching the time-price table of $J_{jl^*}$ (i.e., Table I), the *Lookup function* can obtain the costs of the machines indexed by its second and third arguments. Each time the next faster machine (i.e., $u-1$) is selected when more $\delta_{jl^*}$ is paid. The final distribution information is updated in the profile variables (Line 18-28).

*Theorem 4.1:* Given budget $B_j$ for stage $j$ having $n_j$ tasks, Algorithm 1 yields the optimal solution to the distribution of the budget $B_j$ to all the $n_j$ tasks in that stage within time $O(\frac{B_j \log n_j}{\min_{0\leq l\leq n_j}\{\delta_{jl}\}} + n_j)$.

*Proof:* Given budget $B_j$ to stage $j$, by following the greedy algorithm, we can obtain a solution $\Delta_j = \{b_0, b_1, ..., b_{n_j}\}$ where $b_l$ is the budget allocated to task $J_{jl}, 0 \leq l \leq n_j$. Based on this sequence, we can further compute the corresponding finish time sequence of the tasks as $t_0, t_1, ..., t_{n_j}$. Clearly, there exists $k \in [0, n_j]$ that determines the stage completion time to be $T_j(B_j) = t_k$

Suppose $\Delta_j^* = \{b_0^*, b_1^*, ..., b_{n_j}^*\}$ is an optimal solution and its corresponding finish time sequence is $t_0^*, t_1^*, ..., t_{n_j}^*$. Given budget $B_j$, there exists $k' \in [0, n_j]$ satisfying $T_j^*(B_j) = t_{k'}^*$. Obviously, $t_k \geq t_{k'}^*$. In the following we will show $t_k = t_{k'}^*$.

To this end, we consider two cases:

1) If for $\forall l \in [0, n_j]$, $t_l^* \leq t_l$, then we have $b_l^* \geq b_l$. This is impossible because given $b_l^* \geq b_l$ for $\forall l \in [0, n_j]$, the greedy algorithm would have sufficient budget $\geq b_k^* - b_k$ to further reduce $t_k$ of $task_k$, which is contradictory to $T_j(B_j) = t_k$, unless $b_k^* = b_k$, but in this case, $T_j^*(B_j)$ will be $t_k$, rather than $t_k^*$ (Again, case 1 is impossible).
2) Given the result in 1), there must exist $i \in [0, n_j]$ that satisfies $t_i < t_i^*$. This indicates that in the process of the greedy choice, $task_i$ is allocated budget to reduce the execution time at least from $t_i^*$ to $t_i$. In other words, $t_i^*$ once determined the stage completion time during the greedy choice process and this happened no later than when $T_j(B_j) = t_k$. Therefore, we have $t_i^* \geq t_k \geq t_k^* \geq t_i^*$, then $t_k = t_{k'}^*$.

Overall, $t_k = t_{k'}^*$, that is, the algorithm making the greedy choice at every step produces an optimal solution.

The time complexity of this algorithm is straightforward. It consists of the overhead in initialization (Lines 6-10) and the main loop to update the profile variables (Lines 11-30). Since the size of $J_j$ is $n_j$, the initialization overhead is $O(n_j)$. If we adopt some advanced data structure to organize $T_{jl}, 0 \leq l \leq n_j$ for efficient identification of the slowest task, $l^*$ can be obtained within $O(\log n_j)$ (Line 12). On the other hand, there are at most $O(\frac{B_j}{\min_{0\leq l\leq n_j}\{\delta_{jl}\}})$ iterations (Line 11). Overall, we have the time complexity of $O(\frac{B_j \log n_j}{\min_{0\leq l\leq n_j}\{\delta_{jl}\}} + n_j)$. ■

Since all the $\kappa$ stages can be computed in parallel, the total time complexity for the parallel pre-computation is $O(\max_{j\in[0,\kappa)}\{\frac{B_j \log n_j}{\min_{0\leq l\leq n_j}\{\delta_{jl}\}} + n_j\})$.

Given Theorem 4.1, we can immediately have the following corollary, which is a direct result of the first case in the proof of Theorem 4.1.

*Corollary 4.2:* Algorithm 1 minimizes the budget to achieve the optimal stage execution time.

*2) Global Distribution:* Now we consider the second step. Given the results of Algorithm 1 for all the $\kappa$ stages, we try to obtain a dynamic programming recursion to compute the global optimal result. To this end, we use $T(j, r)$ to represent the minimum total time to complete stages indexed from $j$ to $\kappa$ when budget $r$ is available, and have the following recursion ($0 < j \leq \kappa, 0 < r \leq B$):

$$T(j,r) = \begin{cases} \min\limits_{0<q\leq r}\{T_j(n_j,q) + T(j+1, r-q)\} & \text{if } j < \kappa \\ T_j(n_j,r) & \text{if } j = \kappa \end{cases} \quad (6)$$

where the optimal solution can be found in $T(1,B)$. The scheduling scheme can be reconstructed from $T(1,B)$ by recursively backtracking the DP matrix in (6) up to the initial budget distribution at stage $\kappa$, which can phase by phase converge to the final optimal result. To this end, in addition to the time value, we can only store the budget $q$ and the index of the previous stage (i.e., $T(j+1, r-q)$) in each cell of the matrix since given the budget for each stage, we can simply use Algorithm 1 to recompute the budget distribution.

*Theorem 4.3:* Given budget $B$ for a $\kappa$-stage MR job, each stage $j$ having $n_j$ tasks, Recursion (6) yields the optimal solution to the distribution of the budget $B$ to all the $\kappa$ stages within the time complexity of $O(\kappa B^2)$ when $T_j(n_j,q), 0 < j \leq \kappa, 0 < q \leq B$ is pre-computed. Otherwise, it would be $O(nB^3)$ if online computed.

*Proof:* We prove this by induction on the number of stages ($\kappa$). Let the number of stages, $\kappa = 1$. Clearly, given budget $r$, the optimal solution is obtained by $T_j(n_j,r)$. Suppose there are $\kappa$ stages. Consider stages $j$ and $j+1$. As an induction hypothesis, let $T(j+1,p)$ be an optimal solution to stages from $j+1$ to $\kappa$ given budget $p$. We will show that $T(j,r)$ is an optimal solution to stages from $j$ to $\kappa$ under budget constraint $r$. In order to find the optimal distribution of the budget $r$ among $\kappa - j + 1$ stages, we need to consider all the possibilities. To this end, we assign $q$ units to the first stage $j$ and the remaining $r-q$ units to the leftover stages from $j+1$ to $\kappa$, and allow $q$ to be varied in the range of $(0,r]$. Clearly, the recursion chooses the minimum of all these, thus serving all the stages from $j$ to $\kappa$ with a minimum time.

Finally, at stage 1, since there are no more previous stage, the recursion (6) yields the optimal result $T(1,B)$ for the workflow. Since there are $O(\kappa B)$ elements in DP matrix (6). For each element, the computation's complexity is at most $O(B)$ when $T_j(n_j,q), 0 < q \leq r$ have been pre-computed. Therefore, the total time complexity is $O(\kappa B^2)$. Otherwise, it would be written as $B(\sum_{j=1}^{\kappa}\sum_{q=0}^{B}(\frac{q\log n_j}{\min_{0\leq l\leq n_j}\{\delta_{jl}\}} + n_j))$ which is upper bounded by $O(nB^3)$ given $n = \sum_{j=1}^{\kappa} n_j$. Hence, the proof. ■

## V. Empirical Studies

To verify the proposed algorithm and study its performance behaviours in reality,[1] we developed a *budget distribution solver* in Java that efficiently implements the algorithm for the problem subject to budget constraints. Since the monetary cost is our primary interest, in the solver we do not consider some properties and features of the network platforms. Rather, we focus on the factors closely related to our research goal.

The solver accepts as an input a bag of MapReduce jobs that are organized as a multi-stage fork&jon workflow by the scheduler at run-time. Each task of the job is associated with a time-price table, which is pre-defined by the cloud providers. As a consequence, the solver can be configured by several parameters including those describe the time-price tables, the number of tasks in each stage and the total number of stages in the workflow. Since there is no well-accepted model to specify these parameters, we allow them to be automatically generated using a uniform distribution. All the experiments are conducted on Ubuntu 12.04 with a hardware configuration of 3392.183 MHz processors, with a total of 8 processors activated, each with 8192K cache.

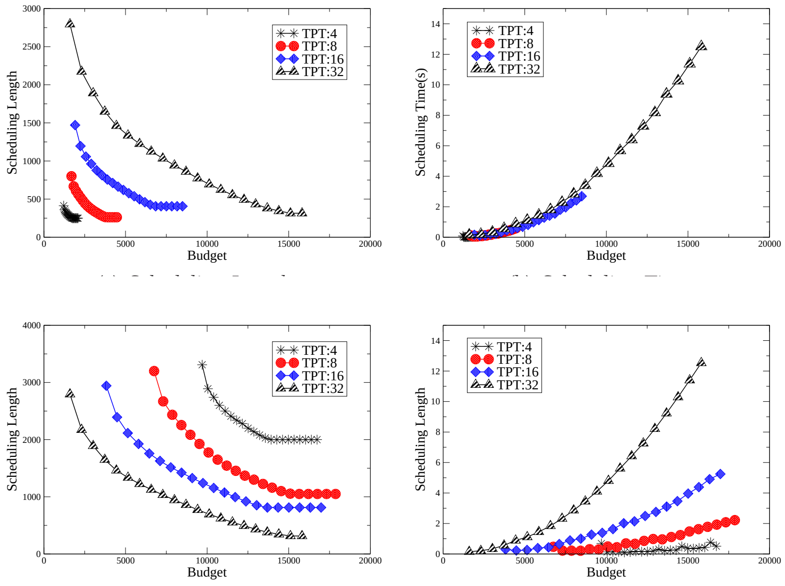
We first evaluate the impact of the budget limits and time-price table sizes on the total scheduling length of the workflow. To this end, we fix an 8-stage workflow with at most 20 tasks in each stage, and the size of the time-price table associated with each task is varied by $4$, $8$, $16$ and $32$. In each table, the task execution time and the corresponding prices are uniformly distributed in [1, 12.5*table_size] and [1, 10*table_size], respectively. Note that this configuration is only designed for observing the optimal algorithm's behavior and verifying its correctness.

The results are shown in Figure 3 where for each trail, the budget is changed from its lower bound to upper bound, which are respectively defined to be the minimal and maximal budget that can be used to complete the workflow. With the budget increasing, for all sizes of the tables, the scheduling lengths are super linearly decreased (Figure 3(a)). This observation is interesting and hard to obtain from the analysis of the algorithm. We attribute this result to the fact that the opportunities of decreasing the execution time of each stage are super linearly increased with budget growth. This phenomenon implies that the ratio *performance/cost* is increased if the cloud users are willing to pay more for the performance.

However, this benefit is not free: the expense is the scheduling time as shown in Figure 3(b) where it dramatically increases with respect to the budget increments. This can be explained from the quadratic time complexity of the algorithm. Therefore, the cloud users should make a careful trade-off between the benefits and the costs.

Another observation is the impact of the table sizes on the performance of the workflows. To make a fair comparison, we scale up the monetary budgets, scheduling lengths and times of each workflow with respect to the largest workflow. For instance, we multiply 8 and the corresponding values of 4-stage workflows to obtain the numbers at same scale with the 32-stage workflow. After the scale-up, the results are shown in Figure 3(c) and (d) where From Figure 3, we can easily see that the performance benefit with respect to a fixed

[1]As our algorithms are claimed to be optimal, it does not fully make sense to compare it with any sub-optimal existing algorithms.



(c) Scaled Scheduling Length (d) Scaled Scheduling Time

Figure 3: Impact of time-price table (TPT) size on the total scheduling length and scheduling time (Stage:8, Task: $\le 20$)

budget is increased as the tables become large. Again, this benefit is at the cost of scheduling time. This result is not surprising as the algorithm makes a global optimization, the larger the workflow is, the more benefit can be obtained. Of course, the time needed is also longer.

Then, we evaluate the performance changes with respect to workflow size when the budget is increased from the lower bound to the upper bound. To this end, we fix the maximum number of tasks as 20 in each stage, and each task has a time-price table with maximum size of 16. We vary the number of stages from $4, 8, 16$ to 32, and observe the performance and time overhead in Figure 4. They exhibit the same behavior patterns of those we achieved when the table sizes are varied. These results are expected as both the number of stages and the size of tables are related to the total workloads in a linear fashion.

Finally, we verify the optimality of the algorithm by using the same set of workflows as in the previous experiments. To this end, by following the same principle of Recursion (6), we design another dynamic programming algorithm as a per-stage distribution algorithm which is shown in Recursion (7) where $T_i[j, b]$ represents the minimum time to complete jobs indexed from $j$ to $n_j$ given budget $b$, in which $0 \le i < \kappa$, $0 \le l \le n_i$, and $0 < b \le B_i$.

$$\begin{cases} T_i[j,b] = \min\limits_{0<q\le b}\{\max\{T_{i\_j}[q], T_i[j+1, b-q]\}\} & \\ T_i[n_i, B_{i\_n_i}] = T_{i\_n_i}(B_{i\_n_i}) & B_{i\_n_i} \le B_i \end{cases} \tag{7}$$

The optimal solution to stage $i$ can be found in $T_i[0, B_i]$. Given the proof of Recursion (6), the correctness of this algorithm is easy to follow. We combine this algorithm with Recursion (6) to achieve the global results of the workloads in our first experiment and compare these results with our current ones. The comparison confirms the optimality of the greedy algorithm.

## VI. Conclusions

In this paper, we addressed two practical constraints, budget and deadline, for the optimal scheduling of a bag of MapReduce jobs on a set of (virtual) machines in Clouds. We presented two parallel optimal algorithms to address these conflicting constraints with pseudo polynomial time complexity. In particular, we focused on the scheduling-length optimization under the budget constraints and designed a greedy algorithm for per-stage budget distribution, which was also shown to be optimal.

Finally, in this study we did not model communication cost since, in cloud platforms, the communication costs are always homogeneous in terms of billing models.

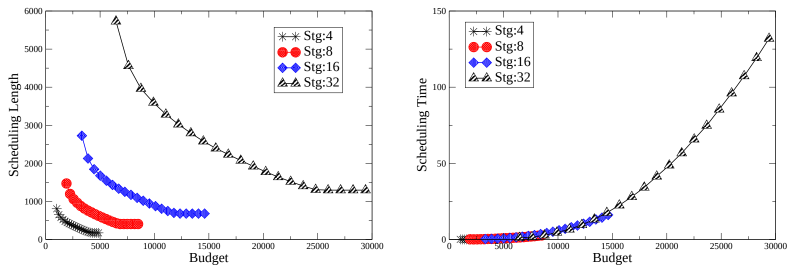
## References

[1] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, dec 2008, pp. 640 –645.

[2] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, "An evaluation of the cost and performance of scientific workflows on amazon ec2," *J. Grid Comput.*, vol. 10, no. 1, pp. 5–21, mar 2012.

[3] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, ser. SC '08, Piscataway, NJ, USA, 2008, pp. 50:1–50:12.

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.



(a) Scheduling Length (b) Scheduling Time

Figure 4: Impact of workflow size on the total scheduling length and scheduling time (TPT:$\leq 16$, Task: $\leq 20$)

[5] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ser. ICDM '08, 2008, pp. 512–521.

[6] Q. Zou, X.-B. Li, W.-R. Jiang, Z.-Y. Lin, G.-L. Li, and K. Chen, "Survey of mapreduce frame operation in bioinformatics," *Briefings in Bioinformatics*, 2013.

[7] H. Liu and D. Orban, "Cloud mapreduce: A mapreduce implementation on top of a cloud operating system," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011, pp. 464–474.

[8] 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*, ser. HPDC '09, 2009, pp. 65–66.

[9] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, "Evaluating mapreduce on virtual machines: The hadoop case," in *Proceedings of the 1st International Conference on Cloud Computing*, ser. CloudCom '09, 2009, pp. 519–528.

[10] M. Correia, P. Costa, M. Pasin, A. Bessani, F. Ramos, and P. Verissimo, "On the feasibility of byzantine fault-tolerant mapreduce in clouds-of-clouds," in *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, 2012, pp. 448–453.

[11] F. Marozzo, D. Talia, and P. Trunfio, "Enabling reliable mapreduce applications in dynamic cloud infrastructures," *ERCIM News*, vol. 2010, no. 83, pp. 44–45, 2010.

[12] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10, 2010, pp. 265–278.

[13] H. Chang, M. Kodialam, R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, "Scheduling in mapreduce-like systems for fast completion time," in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 3074–3082.

[14] H.-H. You, C.-C. Yang, and J.-L. Huang, "A load-aware scheduler for mapreduce framework in heterogeneous cloud environments," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC '11, 2011, pp. 127–132.

[15] B. Thirumala Rao and L. S. S. Reddy, "Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments," *International Journal of Computer Applications*, vol. 34, no. 9, Nov. 2011.

[16] Y. Li, H. Zhang, and K. H. Kim, "A power-aware scheduling of mapreduce applications in the cloud," in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, 2011, pp. 613–620.

[17] P. Kondikoppa, C.-H. Chiu, C. Cui, L. Xue, and S.-J. Park, "Network-aware scheduling of mapreduce framework ondistributed clusters over high speed networks," in *Proceedings of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit*, ser. FederatedClouds '12, 2012, pp. 39–44.

[18] 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*, ser. OSDI'08, 2008, pp. 29–42.

[19] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive - a petabyte scale data warehouse using hadoop," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, 2010, pp. 996–1005.

[20] D. Pisinger, "A minimal algorithm for the multiple-choice knapsack problem." *European Journal of Operational Research*, vol. 83, pp. 394–410, 1994.

[21] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04, 2004, pp. 10–10.

[22] Apache Software Foundation. Hadoop, http:/hadoop.apache.org/core.

[23] 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 SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07, 2007, pp. 59–72.

[24] Greenplum HD, http:/www.greenplum.com.

[25] Hadoop FairScheduler http:/hadoop.apache.org/common/docs/currrent/fair_scheduler.html.

[26] Hadoop CapacityScheduler http:/hadoop.apache.org/common/docs/currrent/capacity_scheduler.html.

[27] Y. Li, H. Zhang, and K. H. Kim, "A power-aware scheduling of mapreduce applications in the cloud," in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, 2011, pp. 613–620.

[28] K. Kc and K. Anyanwu, "Scheduling hadoop jobs to meet deadlines," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 388–392.

[29] K. Wang, B. Tan, J. Shi, and B. Yang, "Automatic task slots assignment in hadoop mapreduce," in *Proceedings of the 1st Workshop on Architectures and Systems for Big Data*, ser. ASBD '11, 2011, pp. 24–29.

[30] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Sci. Program.*, vol. 14, no. 3,4, pp. 217–230, Dec. 2006.

[31] L. Zeng, B. Veeravalli, and X. Li, "Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud," in *Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, ser. AINA '12, 2012, pp. 534–541.

[32] E. Caron, F. Desprez, A. Muresan, and F. Suter, "Budget constrained resource allocation for non-deterministic workflows on an iaas cloud," in *Proceedings of the 12th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ser. ICA3PP'12, 2012, pp. 186–201.