Replicated Data Placement for Uncertain Scheduling

Manmohan Chaubey, Erik Saule
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, USA
Email: mchaubey@uncc.edu, esaule@uncc.edu

***Abstract*—Scheduling theory is a common tool to analyze the performance of parallel and distributed computing systems, such as their load balance. How to distribute the input data to be able to execute a set of tasks in a minimum amount of time can be modeled as a scheduling problem. Often these models assume that the computation time required for each task is known accurately. However in many practical case, only approximate values are available at the time of scheduling.**

In this paper, we investigate how replicating the data required by the tasks can help coping with the inaccuracies of the processing times. In particular, we investigate the problem of scheduling independent tasks to optimize the makespan on a parallel system where the processing times of tasks are only known up to a multiplicative factor. The problem is decomposed in two phases: a first offline phase where the data of the tasks are placed and a second online phase where the tasks are actually scheduled.

For this problem we investigate three different strategies, each allowing a different degree of replication of jobs: a) No Replication b) Replication everywhere and c) Replication in groups. We propose approximation algorithms and theoretical lower bound on achievable approximation ratios. This allows us to study the tradeoff between the number of replication and the guarantee on the makespan.

***Keywords*-Scheduling; Uncertainty; Robustness; Replication; Approximation Algorithms; Parallel System**

I. Introduction

Parallel and distributed computing systems are often modeled using tasks that are processed simultaneously on different machines. Studying the balance of the load of the various component of the system is often key in understanding the performance one obtains in practice. A system typically schedules the set of tasks with the goal of optimizing the load balance (or makespan) of the system or some other metric. A key information these system use to plan the execution is the time tasks will take to be processed. However, this information is typically not precisely known in practice: because the user can only make a wild guess on the runtime of her task [16], because prediction is hard in the general case [24], or because underlying models of a particular algorithm can only predict runtime within a given range [7]. Whichever the reason is, not knowing accurately the processing time can significantly impact the performance obtained from the machine.

Though, the uncertainty in processing time is only a real problem if the task is pinned to a particular computing unit. If the task could be moved as the system sees fit without incurring an extra cost, the problem would be vastly alleviated. The problem is that in practice, the task has to run on a particular machine because the input data for this task are only available on this machine. This is the case in most parallel and distributed systems and it is particularly true in out-of-core computing systems. For instance in out-of-core sparse linear algebra, executing a task where the data are not locally available would have a prohibitive overhead [28], [27].

One approach for dealing with the uncertainty of processing time is to build a robust schedule [2], [9], [6], that is, building a schedule that can naturally cope with variations in the processing times. These techniques often use sensitivity analysis to determine the robustness of the schedule. However, a better approach would be to be able to dynamically change the schedule.

The idea we are pursuing in this paper is to replicate the input data of the tasks onto multiple machines. This way, when the actual processing times of the tasks are too different from their estimations, the system will have some room to adapt at runtime. This is certainly feasible in practice as many system have more memory than the computation use. For instance, most Hadoop system replicates the data for the purpose of tolerating hardware faults [22]. And it has been shown that launching the same task multiple times can help cope with hardware differences [21] but increases resource usage. The cost of replicating the data might be amortized in many applications where the application will iterated over the data multiple times (*e.g.*, in an iterative solver [27], [28]). In this paper, we answer the question "can data replication help cope with the uncertainty of processing time?" And the answer is that it can.

The remaining of the paper is organized as follows:

we describe system model and notations in Section II. Related works are presented in Section III. Section IV investigates what can be done if the tasks can only be deployed on a single machine, we provide a guaranteed algorithm and provide a lower bound on the best guarantee that one can achieve in this case. Section V takes the reverse case and investigates what can be achieved if the data are replicated everywhere, leaving the maximum flexibility at runtime. We investigate one algorithm in this case and analyze its performance guarantee. Section VI investigates grouping processors together and replicating data in these groups as an intermediate between the previous two strategies and provide a guaranteed algorithm in that case. Section VII summarizes the various results we derived and study the tradeoff between performance guarantee and data replication.

II. Problem Definition

Let J be a set of n jobs which need to be scheduled onto a set M of m machines. We will use interchangeably the terms machines and processors. Also we will use interchangeably the terms jobs and tasks. We are considering the problem where the scheduler does not know the processing time p_j of task j exactly before the task completes. But the scheduler has access to some estimation of the processing time $\tilde{p_j}$ of task j before making any scheduling decisions. We assume that the actual processing time p_j of a task j is within a multiplicative factor α of the estimated processing time $\tilde{p_j}$. α is a quantity known to the scheduler. In other words the scheduler knows that:

$$\frac{\tilde{p}_j}{\alpha} \leq p_j \leq \alpha \tilde{p}_j \qquad (1)$$

Assuming that the processing time of the tasks is known to be in an interval is reasonable in many application scenarios. One could derive bounds experimentally using machine learning techniques: for instance [26] used Support Vector Machines to predict the time it will take to run graph traversal algorithms. Models of runtime of algorithms can also be derived analytically: in [7] the authors provide bounds for the performance of various sparse linear algebra operations using only the size of the matrices and vectors involved.

The scheduling for the problem is performed in two phases. Phase 1 chooses where data are replicated using the estimated processing time $\tilde{p_j}$, for each of the task j. The phase takes $\tilde{p_j}$, m and α as inputs and outputs sets of machines, $M_j \subseteq M$ where each task j can be scheduled. This phase is purely offline and corresponds to the operations performed to prepare the execution of the application.

Phase 2 takes the output of phase 1 as its input and maps each task j to a machine within the set of machines M_j. For each machine i, let $E_i \subseteq J$ be the set of tasks assigned to machine i. This phase chooses the actual schedule following an an online semi-clairvoyant process. Only the approximate processing time is known when a task is placed, but the scheduler can wait for a machine to become idle, to place the next one. Therefore, can dynamically schedule the tasks and the actual processing time of the tasks are known once they complete.

The problem is to optimize the makespan, $C_{max} = \max_i \sum_{j \in E_i} p_j$ which is the completion time of the last task of the system. C^*_{max} denotes the optimal makespan of an instance of the problem (knowing the actual processing times). An offline algorithm is said to be a ρ-approximation algorithm (or to have an approximation ratio of ρ) if it guarantees for all the instances that $C_{max} \leq \rho C^*_{max}$. When the problem is online, we are talking about competitive ratios.

III. Related Work

When $\alpha = 1$, the problem is exactly the classical independent tasks scheduling problem on identical machines, which is known to be NP-Hard [8]. We use Graham's List Scheduling (LS) [10] and Largest Processing Time (LPT) algorithms [11] to derive approximation ratios in different scenarios. The LS algorithm takes tasks one at a time and assigns them to the processor having the least load at that time. LS is a 2-approximation algorithm and is widely used in online scheduling problems. LPT sorts the tasks in a non-increasing order of processing time and assigns them one at a time in this order to the processor with the smallest current load. The LPT algorithm has a worst case approximation ratio of $\frac{4}{3} - \frac{1}{3m}$ in the offline setting. One can even obtain an arbitrarily good approximation algorithm for this problem by increasing its complexity with a dual approximation algorithm [12].

Based on various models for describing the uncertain input parameter, various methodologies can be used including reactive, stochastic, fuzzy and robust approach [14]. We are using the bounded uncertainty model which assumes that an input parameter have value between a lower and upper bound. Wierman and Nuyens [23] introduce SMART, a classification to understand size-based policies and draw analytic corelation between response time and estimated job size in single server problem. Robust approaches to deal with uncertainty are widely used on MapReduce systems [13] [20], in Hadoop [25] [22], on databases [15] and on web servers [3]. The HSFS and FLEX schedulers provide robustness in scheduling against uncertain job size [25], [17]. Cannon and Jeannot [2] analyzed the correlation between various metrics used to measure robustness and provided scheduling heuristics that optimizes both makespan and robustness for scheduling task graph on heterogeneous system.

Most of the work on robust scheduling use scenarios to structure the variability of uncertain parameters. Daniels and Kouvelis [5] used them to optimize the flow-time using a single machine. Davenport, Gefflot, and Bek analyzed slack based technique (adding extra idle time) to cope with uncertainty [6]. Gatto and Widmayer derives bounds on competitive ratio of Graham's online algorithm in scenario where processing times of jobs either increase or decrease arbitrarily due to perturbations [9]. These works considered augmenting or decreasing of job processing times as different problem scenario that need to be optimized. We approach the problem using worst case analysis where some tasks may increase and some may decrease within the same schedule.

Data placement and replication methodologies are highly used in distributed systems including peer-to-peer and Grid systems to achieve effective data management and improve performance [4][1][18]. Tse [19] used selective replication of documents to increase the available bandwidth to serve files using web servers and study the problem through bi-criteria optimization techniques to maximize the quality of service and minimize the memory occupation.

IV. Strategy 1: No Replication

This section considers the situation where the data of each task is restricted to be on only one machine, *i.e.*, $\forall j, |M_j| = 1$. We have a set J of n jobs, and a set M of m machines. Let $f : J \mapsto M$ be a function that assigns each job to exactly one machine. The restriction that the data of each task is deployed on a single machine puts all the decision in phase 1: each task can only be scheduled on one machine in phase 2.

A. Lower Bound

Theorem 1. *When $|M_j| = 1$, there is no online algorithm having competitive ratio better than $\frac{\alpha^2 m}{\alpha^2+m-1}$.*

Proof: We use the adversary technique to prove the lower bound of this theorem. An adversary discloses the input instance piece by piece. It analyzes the choices made by the algorithm to change the part of the instance that has not been disclosed yet. That way it can build an instance that maximizes the competitive ratio of the algorithm.

Let us consider an instance with λm tasks of equal estimated processing time $\forall j, \tilde{p_j} = 1$. After phase 1, let i be the most loaded processor which has B tasks. Obviously, $B \geq \lambda$. In phase 2 the adversary increases the processing time of the tasks on processor i by a factor of α and changes the processing time of the other tasks by a factor of $\frac{1}{\alpha}$. So, $C_{max} = \alpha B$. C^*_{max} will be no worse than any feasible solution. In particular, the solution that distributes equally the jobs of size α and the jobs of size $\frac{1}{\alpha}$. Therefore $C^*_{max} \leq \frac{1}{\alpha}\lceil\frac{\lambda m - B}{m}\rceil + \alpha\lceil\frac{B}{m}\rceil$. Figure 1 depicts the online solution and the offline optimal. We have,

$$\frac{C_{max}}{C^*_{max}} \geq \frac{\alpha^2 B}{\lceil\frac{\lambda m-B}{m}\rceil + \alpha^2\lceil\frac{B}{m}\rceil}$$

Since $\frac{\lambda m-B}{m} + 1 \geq \lceil\frac{\lambda m-B}{m}\rceil$ and $\frac{B}{m} + 1 \geq \lceil\frac{B}{m}\rceil$, we have

$$\frac{C_{max}}{C^*_{max}} \geq \frac{\alpha^2 B}{\frac{\lambda m-B}{m} + 1 + \alpha^2\frac{B}{m} + \alpha^2}$$

From above expression it is clear that the smaller the value of B, the more the value of the expression decreases. So, any algorithm should minimize B to achieve better performance. For a schedule to be feasible the condition $B \geq \lambda$ must be satisfied. For $B = \lambda$ the value of $\frac{C_{max}}{C^*_{max}}$ is minimum and is equal to $\frac{\alpha^2 m\lambda}{\lambda(\alpha^2+m-1)+m(\alpha^2+1)}$. The adversary can maximize the ratio of the algorithm by arbitrarily increasing λ. When λ tends to ∞, we have

$$\frac{C_{max}}{C^*_{max}} \geq \frac{\alpha^2 m}{\alpha^2+m-1}$$

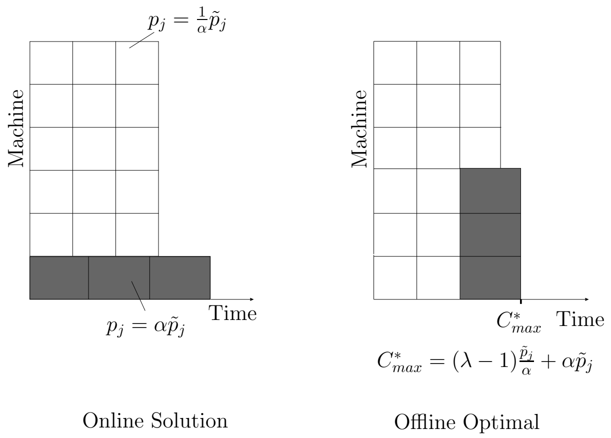
∎

Corollary 1.1. *When m goes to ∞ there is no online algorithm having competitive ratio better than α^2.*

B. Algorithm

We present the algorithm **LPT-No Choice**. In phase 1, the algorithm distribute the data of the tasks to the processor using their estimated processing times according to Graham's LPT algorithm [11]: The tasks are sorted in non-increasing order of their processing time and are greedily scheduled on the processor that minimizes the sum of $\tilde{p_j}$ of the tasks allocated on that processor. Since there is no replication, there is no decision to take in phase 2.

The performance of the algorithm depends mostly on how much the actual processing times of the tasks differ from their estimation. It also depends on the existence of a better arrangement if the actual processing times were known. The following theorem states the theoretical guarantee of the algorithm.



$p_j = \frac{1}{\alpha}\tilde{p_j}$
Machine
Time
$p_j = \alpha\tilde{p_j}$
Online Solution
Machine
C^*_{max}
Time
$C^*_{max} = (\lambda-1)\frac{\tilde{p_l}}{\alpha} + \alpha\tilde{p_j}$
Offline Optimal

Figure 1: Instance constructed by the adversary in the proof of theorem 1 with $\lambda = 3$ and $m = 6$. In the online solution, the adversary increases the processing time of a task of the most loaded machine by a factor of α. If that information was available beforehand, an optimal offline algorithm could have distributed these longer tasks to other processors.

Theorem 2. *The* ***LPT-No Choice*** *has a competitive ratio of* $\frac{2\alpha^2 m}{2\alpha^2+m-1}$.

Proof: The algorithm assigns the jobs to processors based on their estimated processing times using LPT in Phase 1. So, the planned makespan considering the estimated processing times of tasks, $\tilde{C}_{max}$ have the following relation with the total estimated processing time, $\tilde{p_j}$ and estimated processing time of the task l that reaches $\tilde{C}_{max}$.

$$\tilde{C}_{max} \leq \frac{\sum \tilde{p_j} + (m-1)\tilde{p_l}}{m} \tag{2}$$

The actual makespan of a schedule, C_{max}, obtained using the actual processing times of all the jobs, must be smaller than $C_{max} \leq \alpha\tilde{C}_{max}$ (thanks to Equation 1). We have following inequality:

$$C_{max} \leq \alpha\tilde{C}_{max} \leq \alpha\left(\frac{\sum \tilde{p_j} + (m-1)\tilde{p_l}}{m}\right) \tag{3}$$

The worst case situation is when the task of the processor where the sum of estimated processing time is $\tilde{C}_{max}$ sees the actual processing time of its task being α times larger than their estimate; meanwhile the processing time of the task on the rest of the processors is $\frac{1}{\alpha}$ times their estimation. The argument behind this statement is that greater the value of ratio $\frac{C_{max}}{\sum p_j}$, the worse the algorithm approximation ratio will be. So the total actual processing time is given by the following equation.

$$\sum p_j = \frac{\sum \tilde{p_j} - \tilde{C_{max}}}{\alpha} + \alpha\tilde{C}_{max} \tag{4}$$

Also the actual optimal makespan have following constraint

$$C^*_{max} \geq \frac{\sum p_j}{m}$$

Substituting for $\sum p_j$, we have

$$mC^*_{max} \geq \frac{\sum \tilde{p_j} - \tilde{C_{max}}}{\alpha} + \alpha\tilde{C}_{max}$$

$$mC^*_{max} \geq \frac{\sum \tilde{p_j} - \left(\frac{\sum \tilde{p_j} + (m-1)\tilde{p_l}}{m}\right)}{\alpha} + C_{max}$$

$$mC^*_{max} \geq \frac{m-1}{\alpha m}\left(\sum \tilde{p_j} - \tilde{p_l}\right) + C_{max}$$

By the property of LPT, $\sum \tilde{p_j} - \tilde{p_l} \geq m(\tilde{C}_{max} - \tilde{p_l})$, we have,

$$mC^*_{max} \geq \frac{m-1}{\alpha}\left(\tilde{C_{max}} - \tilde{p_l}\right) + C_{max}$$

All instances where there is only one task per processor is always optimal. Therefore, we can restrict our analysis without loss of generality to instances with at least two jobs per processor. (Notice that in the original proof of Graham's LPT [11], an argument is made that all instances with two tasks per machine are optimal. However, the argument does not port in our case where only estimated processing times are known.) For at least two jobs on the processor that reaches $\tilde{C}_{max}$, the (estimated) processing time of last job is smaller than half the estimated makespan, $\tilde{p_l} \leq \frac{\tilde{C}_{max}}{2}$. Substituting this expression in the above equation, we have

$$mC^*_{max} \geq \frac{m-1}{\alpha}\left(\tilde{C}_{max} - \frac{\tilde{C}_{max}}{2}\right) + C_{max}$$

Using equation 3,

$$mC^*_{max} \geq \frac{m-1}{2\alpha}\frac{C_{max}}{\alpha} + C_{max}$$

$$mC^*_{max} \geq \left(\frac{m-1}{2\alpha^2} + 1\right)C_{max}$$

$$\frac{C_{max}}{C^*_{max}} \leq \frac{2\alpha^2 m}{2\alpha^2+m-1}$$

■

V. Strategy 2: replicate data everywhere

With this strategy, we put no restriction on phase 2. The tasks are replicated everywhere i.e. $\forall j, |M_j| = |M|$. We introduce the **LPT-No Restriction** which replicates the data of all the tasks on each machine in the first phase. In the second phase we simply use the Longest Processing Time algorithm (LPT) in an online fashion using the estimated processing time of the task. That is to say, the tasks are sorted in non-increasing order of their estimated processing time. Then the task are greedily allocated on the first processor that becomes available. Note that this is done in phase 2, the processor become available with when the actual processing time of the task scheduled onto it elapse.

Lemma 3. *Let l be the task that reaches C_{max} in the solution constructed by **LPT-No Restriction**. If there are at least two tasks on the machine that executes l in **LPT-No Restriction**, then $C^*_{max} \geq \frac{2}{\alpha^2} p_l$.*

Proof: Since there are at least two tasks on the machine that executes l in **LPT-No Restriction**, there are at least $m+1$ tasks i such that $\tilde{p_j} \geq \tilde{p_l}$. Therefore in any solution at least one machine gets two tasks c and d, such that $\tilde{p}_c \geq \tilde{p}_l$ and $\tilde{p}_d \geq \tilde{p}_l$. C^*_{max} must be greater than sum of the processing time of these two tasks.

$$C^*_{max} \geq p_c + p_d$$

As the actual processing time of a task must be greater than $\frac{1}{\alpha}$ times of its estimated value, we have $p_c \geq \frac{1}{\alpha}\tilde{p}_c$ and $p_d \geq \frac{1}{\alpha}\tilde{p}_d$. Using this

$$C^*_{max} \geq \frac{1}{\alpha}\tilde{p}_c + \frac{1}{\alpha}\tilde{p}_d \geq \frac{2}{\alpha}\tilde{p}_l$$

Since, $\tilde{p}_l \geq \frac{1}{\alpha} p_l$, we have

$$C^*_{max} \geq \frac{2}{\alpha^2} p_l$$

■

Theorem 4. ***LPT-No Restriction** has a competitive ratio of $\frac{C_{max}}{C^*_{max}} \leq 1 + (\frac{m-1}{m})\frac{\alpha^2}{2}$*

Proof: The optimal makespan, C^*_{max} must be at least equal to the average load on the m machines. We have

$$C^*_{max} \geq \frac{\sum p_j}{m} \tag{5}$$

By the property of LPT (actually, it is a property of List Scheduling which LPT is a refinement of) the load on each machine i is greater than the load on the machine which reach C_{max} before the last task l is scheduled. So for each machine i, $C_{max} \leq \sum_{j \in E_i} p_j + p_l$ holds true. Summing for all the machines we have

$$mC_{max} \leq \sum p_j + (m-1)p_l$$

$$C_{max} \leq \frac{\sum p_j}{m} + \frac{(m-1)}{m} p_l \tag{6}$$

Using 5 and 6, we have

$$\frac{C_{max}}{C^*_{max}} \leq 1 + \frac{m-1}{m}\left(\frac{p_l}{C^*_{max}}\right)$$

Using Lemma 3, we have

$$\frac{C_{max}}{C^*_{max}} \leq 1 + \left(\frac{m-1}{m}\right)\frac{\alpha^2}{2}$$

■

Graham's List Scheduling algorithm always has a competitive ratio of $2 - \frac{1}{m}$. For $\alpha^2 < 2$, the **LPT-No Restriction** algorithm has better approximation than List Scheduling. For $\alpha^2 > 2$ List Scheduling has better guarantee than the one expressed in Theorem 4. Since **LPT-No Restriction** is a variant of List Scheduling, the algorithm has a competitive ratio of $\min(1 + \frac{m-1}{2m}\alpha^2, 2 - \frac{1}{m})$.

VI. Strategy 3: Replication in groups

This strategy partitions the processors into k groups $G1, G2...Gk$. The size of each group is equal and have $\frac{m}{k}$ processors within each group. For the sake of simplicity, we assume that we will only use values of k such that k divides m. In the first phase, the data of each task is replicated on all the processors of one of the k groups, i.e. $\forall j, |M_j| = \frac{m}{k}$. In the second phase the tasks are scheduled within the group they are assigned to in first phase. Figure 2 shows the construction of two phases.

We propose the **LS-Group** algorithm which is based on Graham's List Scheduling algorithm. In phase 1, we use List Scheduling to distribute the tasks to the k groups of processors. In phase 2 each task is scheduled to a particular processor within the group it was allocated in phase 1 using the online List Scheduling algorithm.

Theorem 5. *With k groups, the competitive ratio of **LS-Group** is $\frac{k\alpha^2}{\alpha^2+k-1}(1 + \frac{k-1}{m}) + \frac{m-k}{m}$*

Proof: We assume without loss of generality that C_{max} comes from group $G1$. C^*_{max} must be greater than the average of the loads on the machines.

$$C^*_{max} \geq \frac{\sum_{j \in J} p_j}{m}$$

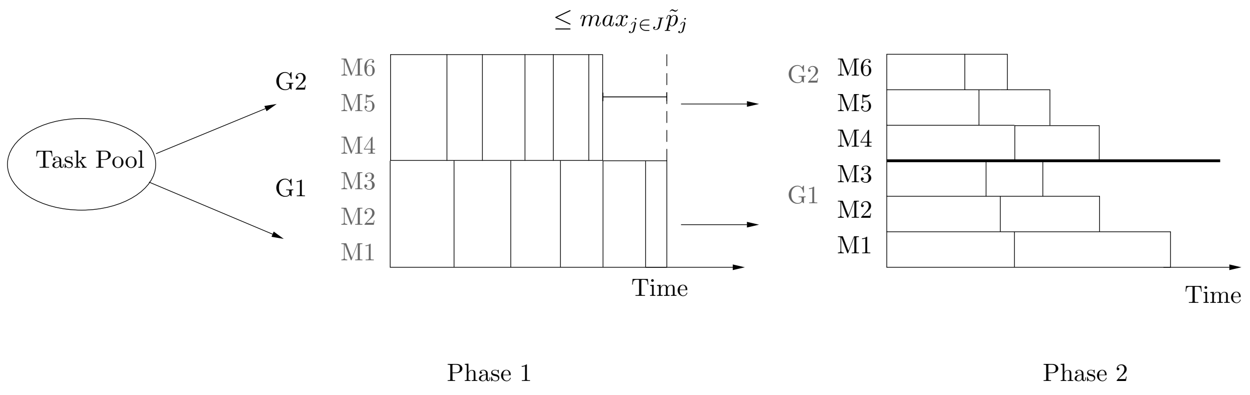


Figure 2: An example of replication in groups with $m = 6$, $k = 2$. In phase 1, the data of the tasks are assigned to one of the groups. Phase 2 schedules each task assigned to a machine within its group.

$\sum_{j\in J} p_j$ can be written as sum of load on $G1$ and load on rest of groups.

$$C^*_{max} \geq \frac{\sum_{j\in G1} p_j + \sum_{l=2}^{k} \sum_{j\in Gl} p_j}{m} \tag{7}$$

As in phase 1 tasks are allocated to different groups using List Scheduling with the estimated processing times of the tasks, the (estimated) load difference between any two groups cannot be greater than the estimated value of largest task $max_{j\in J}\tilde{p}_j$. So, for any group $Gl \neq G1$, We have

$$\forall l \in \{2, 3, \ldots, k\}, |\sum_{j\in G1} \tilde{p}_j - \sum_{j\in Gl} \tilde{p}_j| \leq max_{j\in J}\tilde{p}_j$$

Adding for all values of l leads to

$$|(k-1)\sum_{j\in G1} \tilde{p}_j - \sum_{l=2}^{k}\sum_{j\in Gl} \tilde{p}_j| \leq (k-1)max_{j\in J}\tilde{p}_j$$

Case 1: If $(k-1)\sum_{j\in G1} \tilde{p}_j > \sum_{l=2}^{k}\sum_{j\in Gl} \tilde{p}_j$.

$$\sum_{l=2}^{k}\sum_{j\in Gl} \tilde{p}_j \geq (k-1)\left(\sum_{j\in G1} \tilde{p}_j - max_{j\in J}\tilde{p}_j\right)$$

As the actual processing time of the tasks can vary within a factor α and $\frac{1}{\alpha}$ of their estimated processing time, the following inequality holds

$$\alpha\sum_{l=2}^{k}\sum_{j\in Gl} p_j \geq (k-1)\left(\frac{1}{\alpha}\sum_{j\in G1} p_j - \alpha max_{j\in J}p_j\right)$$

$$\sum_{l=2}^{k}\sum_{j\in Gl} p_j \geq (k-1)\left(\frac{1}{\alpha^2}\sum_{j\in G1} p_j - max_{j\in J}p_j\right) \tag{8}$$

Phase 2 applies List Scheduling in the online mode. We assumed that C_{max} comes from $G1$. Using the guarantees of List Scheduling we can write,

$$C_{max} \leq \frac{\sum_{j\in G1} p_j}{m/k} + \frac{m/k-1}{m/k}p_{max} \tag{9}$$

where p_{max} is actual processing time of longest task in $G1$.

From Equation 8 and 7, we derive

$$C^*_{max} \geq \frac{\sum_{j\in G1} p_j + (k-1)\left(\frac{1}{\alpha^2}\sum_{j\in G1} p_j - max_{j\in J}p_j\right)}{m}$$

$$\alpha^2(mC^*_{max} + (k-1)max_{j\in J}p_j) \geq (\alpha^2+k-1)\sum_{j\in G1} p_j$$

$$\frac{\alpha^2}{\alpha^2+k-1}(mC^*_{max} + (k-1)max_{j\in J}p_j) \geq \sum_{j\in G1} p_j \tag{10}$$

Using 9 and 10, We have

$$C_{max} \leq \frac{k\alpha^2}{\alpha^2+k-1}\left(C^*_{max} + \frac{k-1}{m}max_{j\in J}p_j\right) + \frac{m/k-1}{m/k}p_{max}$$

As $C^*_{max} \geq max_{j\in J}p_j \geq p_{max}$, we have

$$C_{max} \leq \frac{k\alpha^2}{\alpha^2+k-1}\left(C^*_{max} + \frac{k-1}{m}C^*_{max}\right) + \frac{m-k}{m}C^*_{max}$$

So, in Case 1 the algorithm has a competitive ratio of,

$$\frac{C_{max}}{C^*_{max}} \leq \frac{k\alpha^2}{\alpha^2+k-1}\left(1 + \frac{k-1}{m}\right) + \frac{m-k}{m}$$

Case 2: If $(k-1)\sum_{j\in G1}\tilde{p}_j \le \sum_{l=2}^{k}\sum_{j\in Gl}\tilde{p}_j$.

Since the processing times of the tasks can vary within a factor α and $\frac{1}{\alpha}$ of their estimated values, the expression for case 2 can be written as

$$\sum_{l=2}^{k}\sum_{j\in Gl} p_j \ge \frac{1}{\alpha^2}(k-1)\sum_{j\in G1} p_j$$

Putting this value in Equation 7, we have

$$C^*_{max} \ge \frac{\alpha^2+k-1}{m\alpha^2}\sum_{j\in G1} p_j \qquad (11)$$

Using Equations 9 and 11, and as $C^*_{max} \ge p_{max}$, we have

$$C_{max} \le \frac{k\alpha^2}{\alpha^2+k-1}C^*_{max} + \frac{m-k}{m}C^*_{max}$$

So, in case 2 the algorithm has a competitive ratio of $\frac{k\alpha^2}{\alpha^2+k-1} + \frac{m-k}{m}$.

Clearly, the algorithm has a worst competitive ratio in case 1. So, the algorithm has a competitive approximation ratio of $\frac{C_{max}}{C^*_{max}} \le \frac{k\alpha^2}{\alpha^2+k-1}\left(1+\frac{k-1}{m}\right)+\frac{m-k}{m}$. ■

LS-Group uses List Scheduling in both its phases. A LPT-based algorithm may have better guarantee. But without performing any replication, *i.e.* when $k = m$, the **LS-Group** algorithm has a competitive ratio almost equal to **LPT-No choice**'s when the number of machines m is large and the value of α is within practical range. This indicates an LPT-based algorithm for strategy 3 would likely not have a much more interesting guarantee.

VII. Summary

Table 3 summarizes the results of this paper in term of approximation theory. Based on adversary technique, Theorem 1 states that there is no algorithm which can give performance better than $\frac{\alpha^2 m}{\alpha^2+m-1}$ for the model where no replication is allowed. **LPT-No Choice** is a $\frac{2\alpha^2 m}{2\alpha^2+m-1}$-approximation that uses that strategy. For the second strategy that replicates the data of all tasks everywhere ($|M_j| = |M|$), **LPT-No Restriction** achieves a competitive ratio of $1+(\frac{m-1}{m})\frac{\alpha^2}{2}$. The third strategy uses replication within k groups of size m/k (*i.e.*, $|M_j| = m/k$). Using this strategy, the **LS-Group** algorithm has a competitive ratio of $\frac{k\alpha^2}{\alpha^2+k-1}\left(1+\frac{k-1}{m}\right)+\frac{m-k}{m}$.

Of course, there is an inherent tradeoff between replicating data and obtaining good values for the makespan.

Replication	Approximation ratio
$\lvert M_j\rvert = 1$	$\frac{C_{max}}{C^*_{max}} \le \frac{2\alpha^2 m}{2\alpha^2+m-1}$ (Th. 2) No approximation better than $\frac{\alpha^2 m}{\alpha^2+m-1}$ (Th. 1)
$\lvert M_j\rvert = m$	$\frac{C_{max}}{C^*_{max}} \le 1+(\frac{m-1}{m})\frac{\alpha^2}{2}$ (Th. 4) $\frac{C_{max}}{C^*_{max}} \le 2-\frac{1}{m}$ [10]
$\lvert M_j\rvert = \frac{m}{k}$	$\frac{C_{max}}{C^*_{max}} \le \frac{k\alpha^2}{\alpha^2+k-1}\left(1+\frac{k-1}{m}\right)+\frac{m-k}{m}$ (Th. 5)

Table 3: Summary of the contribution of this paper. Three proposed algorithms have guaranteed performance. One lower bound on approximability has been established.

To better understand the tradeoff we show in Figure 4 how the expressions of the guarantees (or impossibility) translate to actual values in a approximation ratio / replication space. We picked 3 values of α while keeping the number of machines fixed $m = 210$.

When $\alpha = 1.1$, even with multiple groups **LS-Group** provides little improvement over **LPT-No Choice**. However there is a significant gap between the guarantee of **LPT-No Choice** and the lower bound on possible approximation. When α is small, there is a significant improvement in using **LPT-No Restriction** over using simply **LS-Group** with only 1 group.

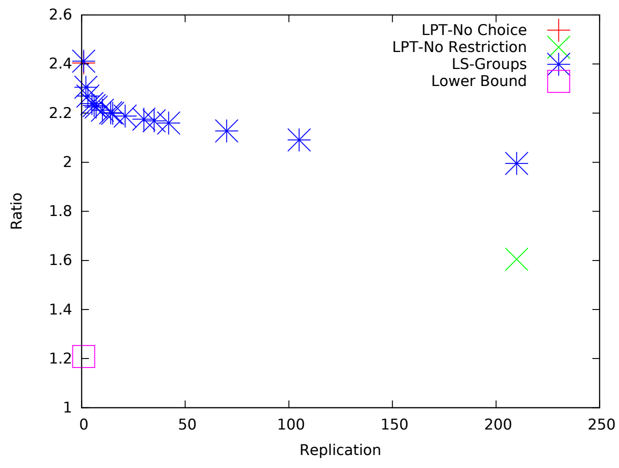
When α increases to 1.5, there is no more differences in the guarantees of **LS-Group** with 1 group and **LPT-No Restriction**. Also **LS-Group** provides many intermediate solution between deploying the data on a single machine and deploying them everywhere.

When $\alpha = 2$, the range of the approximation ratios increase and the value of the lower bound increases. Now **LS-Group** is able to get a better approximation using less than 50 replications than is possible by deploying data on a single machine. Also, the approximation ratio quickly improves from more than 7.5 with the data being replicated on 1 machine to a ratio of less than 6 with only replicating the data on 3 machines.

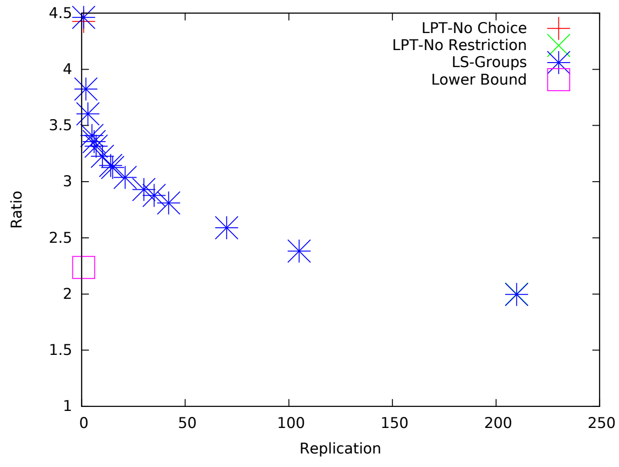
Overall, when α is large, only few replication improve the performance significantly.

VIII. Conclusion and Future Work

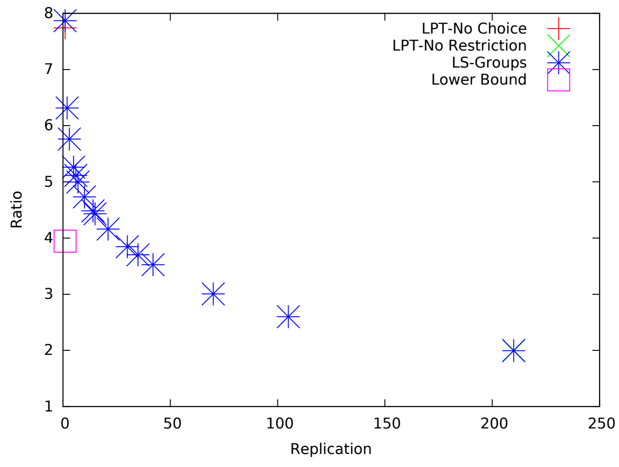
We studied the effect on uncertainty in the processing time of tasks on scheduling for parallel and distributed machines. In particular, we investigated how allowing tasks to execute on different machines can help dealing with not knowing the processing time of tasks accurately. We investigated three different replication strategies, provided approximation algorithm in each case and a lower bound on the best achievable approximation in one of the case. The various strategies allow to trade the number of replication for a better guarantee. In



(a) $m = 210$, $\alpha = 1.1$



(b) $m = 210$, $\alpha = 1.5$



(c) $m = 210$, $\alpha = 2$

Figure 4: Ratio-Replication graph with $m = 210$ and $\alpha \in \{1.1, 1.5, 2\}$.

particular, we obtained a better guarantee with fewer replication than can be achieved by putting the data of a task on only one machine. We observed that even a small amount of replications can improve the guarantee significantly. Therefore, we concluded that deploying the data on multiple machines can be an effective way of dealing with processing time uncertainties.

There are some open problems which can be explored further. Better lower bounds might help understanding the problem better: clearly when α is low, the problem is no different than the offline problem, and when it is large, the problem converges to the non-clairvoyant online problem. Having a clearer idea of where the boundary is will certainly prove useful in understanding how much can be gained using data replication. Also, while replicating data using groups of processor proved effective, more general replication policies can certainly lead to better guarantees.

Though, we assumed that every tasks were replicated the same amount of time. A more realistic model would introduce a cost of replicating a task (either global or per machine). This would allow to replicate only some critical tasks and limit memory usage.

References

[1] J. Abawajy. Placement of file replicas in data grid environments. In M. Bubak, G. van Albada, P. Sloot, and J. Dongarra, editors, *Computational Science - ICCS 2004*, volume 3038 of *Lecture Notes in Computer Science*, pages 66–73. 2004.

[2] L.-C. Canon and E. Jeannot. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, 2010.

[3] V. Cardellini, R. I, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3:28–39, 1999.

[4] W. Cirne, F. Brasileiro, D. Paranhos, L. F. W. Góes, and W. Voorsluys. On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Computing*, 33(3):213 – 234, 2007.

[5] R. L. Daniels and P. Kouvelis. Robust Scheduling to Hedge Against Processing Time Uncertainty in Single-Stage Production. *Management Science*, 41(2):363–376, Feb. 1995.

[6] A. J. Davenport, C. Gefflot, and J. C. Beck. Slack-based techniques for robust schedules. *Proceedings of the Sixth European Conference on Planning (ECP)*, 2001.

[7] G. Erlebacher, E. Saule, N. Flyer, and E. Bollig. Acceleration of derivative calculations with application to radial basis function - finite-differences on the Intel MIC architecture. In *Proc. of International Conference on Supercomputing (ICS)*, 2014.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

[9] M. Gatto and P. Widmayer. On the robustness of graham's algorithm for online scheduling. In *Proc of WADS*, 2007.

[10] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[11] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal On Applied Mathematics*, 17(2):416–429, 1969.

[12] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *Journal of ACM*, 34:144–162, 1987.

[13] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 94–103, 2010.

[14] Z. Li and M. G. Ierapetritou. Process scheduling under uncertainty: Review and challenges. *Computers & Chemical Engineering*, 32(4-5):715–727, 2008.

[15] R. Lipton and J. Naughton. Query size estimation by adaptive sampling. *Journal of Computer and System Sciences*, 51(1):18 – 25, 1995.

[16] D. Luong, J. Deogun, and S. Goddard. Feedback scheduling of real-time divisible loads in clusters. *SIGBED Rev.*, 5(2):2:1–2:4, July 2008.

[17] M. Pastorelli, A. Barbuzzi, D. Carra, M. Dell'Amico, and P. Michiardi. Hfsp: Size-based scheduling for hadoop. In *Big Data, 2013 IEEE International Conference on*, pages 51–59, Oct 2013.

[18] R. Rahman, K. Barker, and R. Alhajj. Study of different replica placement and maintenance strategies in data grid. In *Cluster Computing and the Grid CCGRID*, pages 171–178, 2007.

[19] S. S. H. Tse. Online bounds on balancing two independent criteria with replication and reallocation. *IEEE Trans. Computers*, 61(11):1601–1610, 2012.

[20] A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic resource inference and allocation for MapReduce environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 235–244, New York, NY, USA, 2011. ACM.

[21] D. Wang, G. Joshi, and G. W. Wornell. Efficient task replication for fast response times in parallel computation. In *proc. of SIGMETRICS*, 2014.

[22] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[23] A. Wierman and M. Nuyens. Scheduling despite inexact job-size information. In *Proc. of SIGMETRICS*, pages 25–36, 2008.

[24] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

[25] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. balmin. FLEX: A slot allocation scheduling optimizer for MapReduce workloads. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, pages 1–20, 2010.

[26] Y. You, D. A. Bader, and M. M. Dehnavi. Designing a heuristic cross-architecture combination for breadth-first search. In *Proc. of the 43rd International Conference on Parallel Processing*, 2014.

[27] Z. Zhou, E. Saule, H. M. Aktulga, C. Yang, E. G. Ng, P. Maris, J. P. Vary, and Ü. V. Çatalyürek. An out-of-core dataflow middleware to reduce the cost of large scale iterative solvers. In *2012 International Conference on Parallel Processing (ICPP) Workshops, Fifth International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2)*, Sept. 2012.

[28] Z. Zhou, E. Saule, H. M. Aktulga, C. Yang, E. G. Ng, P. Maris, J. P. Vary, and Ü. V. Çatalyürek. An out-of-core eigensolver on SSD-equipped clusters. In *Proc. of IEEE Cluster*, Sept. 2012.