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 trade-off between the number of replication and the guarantee on the makespan.

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

I. Introduction

the introduction should be structure as follow.

one paragraph on scheduling as a tool for analyzing and optimizing parallel and distributed systems.Mention that in scheduling often processing time are known. But in practice not so much. Give a few references on the difficulty of predicting runtimes in job scheduler.

In parallel and distributed computing, tasks are processed simultaneously on different machines to minimize time to complete the processing of all the tasks. The scheduling for the set of tasks on a given set of machines are usually determined with a goal to optimize certain parameter, such as makespan. The Scheduling study is also used to provide insight into the performance of parallel and distributed computing systems, such as their load balance. The problem of distributing the input data to execute a set of tasks in a minimum amount of time can be formulated as a scheduling problem. Often these scheduling problems are modeled based on the assumption that the computation time required for each task is known accurately; while, in a more practical scenario, only approximate values for the computation times of the tasks can be determined at the time of scheduling.

we need a paragraph on data placement. essentially: uncertainty would not be a problem if the schedules could be dynamically changed. But in practice, the task has to run somewhere because its input data is there. For instance Hadoop, dooc+laf; but essentially in any parallel code. Even if some middleware support executing the task everywhere it pays an overhead for transferring data.

one approach for the problem is to build robust schedule. define robust schedule and give few quick pointers. What is better is to be able to arrange the schedule at runtime. What we investigate: using data replication to give room to the scheduler to move tasks around. Can it help?

In paragraph that gives document organization. In section foo, we present this. In section bar we present that. This is where contribution should be presented. no need to give formula.

Many real world scheduling problems related to task allocation in parallel machines are uncertain in nature. This fact makes the class of problem known as 'scheduling with uncertainty' or 'robust' scheduling an important and most studied problem in Scheduling. Robustness is the measure of to what extent an algorithm can cope with uncertainties in a scheduling problem. Often in scheduling problems the exact processing time of a job is not known initially at the time of planning for job- allocation to different machines, but they have a range outside which their values cannot lie. The fact gives a basis to estimate the processing times and to take job assignment decision based on these estimated values. The actual processing time of a job can vary

drastically from its estimated one. So, any algorithm have to incur the cost of estimation which impacts its performance negatively. A robust approach deals with uncertainty in input parameters to minimize this cost.

The objective of our research is to study the affect on performance of parallel machine scheduling when jobs are scheduled based on their estimated values of processing times, and to propose scheduling approaches that optimizes the performance in this environment. Data placement and replication techniques can be useful in the scenario of uncertain processing times of tasks. Effective data placement using replication strategy allows better load balancing and hence reduces turnaround job time. This paper is important in the sense that it proposes different models depending upon different scenarios and compares them based on approximation ratio and replication they allow. Replication strategy that allows to place data in a wisely manner offers a faster access to files required by jobs, hence increases the job execution's performance. Replication helps in load balancing but it have cost attached with it as it usually increases resource usage [18]. The paper deals with this problem and chooses the scenario in which replication is beneficial. Thus our research focus on two area of scheduling problem i.e. task uncertainty and replication using data placement. The scheduling problem related to data placement and task allocation is common in heterogeneous systems and often dealt with common approach. This paper draws qualitative analysis of the effect of replication in data placement for uncertain scheduling with inexact processing time of a task.

The remaining of the paper is organized as follows: we describe system model and notations in section II. Related works are touched in section III. Sections IV-VI describes the 3 different strategies- a)No replications, b)replication is done everywhere and c)replication is done within a group. **expand per section**. The respective sections describe derivation of competitive ratios for each of these strategies. Section VII summarizes the results of 3 models based on experimental results.

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 problems where the scheduler does not know the processing time p_i of task i exactly before the task completes. But the scheduler has access to some estimation of the processing time $\tilde{p_i}$ of task i before making any scheduling decisions. We assume that the actual processing time p_i of a task i is within a multiplicative factor α of the estimated processing time $\tilde{p_i}$. α is a quantity known to the scheduler. In other words the scheduler knows that:

$$\frac{\tilde{p}_i}{\alpha} \leq p_i \leq \alpha \tilde{p}_i \tag{1}$$

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 [22] 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 vector involved.

The problem is to optimize the makespan, C_{max}. **Define makespan formally. Probably better if we push that at the end of the section.** Makespan is the load on machine which processes the latest task. C^*_{max} denotes optimal makespan of a schedule S.

Write it so that the problem is defined in two phases, what ever the strategy. the strategy we follow does not change the problem. The scheduling for the problem is defined in two phases. Phase 1 chooses where data to be replicated using estimated processing time $\tilde{p_j}$, for each of the task j. The phase takes $\tilde{p_j}$, m and α as input and outputs set of machines, $M_j \subseteq M$ where a task j can be scheduled.

Phase 2 takes output of phase 1 as its input and maps a task j to a machine within the set of machines M_j to which j was allocated in phase 1. For each machine i phase 2 outputs a set of tasks $E_i \subseteq J$ assigned to the machine i. The Phase chooses actual schedule with semi-clairvoyant algorithm which uses only approximate knowledge of initial processing time, and after scheduling the task the actual p_i is known. With objective to find schedule which minimizes makespan, we investigate greedy algorithms for each of the problem models and prove their competitive ratios. **what we investigate has nothing to do here. This is a problem definition.**

III. Related Work

First with $\alpha = 1$ this is the classical scheduling problem.NP-Hardness. cite garey johnson. We have used Graham's List Scheduling (LS) [9] and Largest Processing Time (LPT) algorithms [10] to derive approximation ratios in different scenarios. The LS algorithm takes tasks one at a time and assign it to the processor having least load at that time. LS is 2-approximation algorithm and is widely used in online

scheduling problems. LPT sorts tasks in decreasing order of processing time and assign them one at a time in this order to the processor with the smallest current load. The LPT algorithm has worst case performance ratio as $\frac{4}{3} - \frac{1}{3m}$ in an offline setting. Depending upon which among these two algorithms suits more for a problem model we have used these algorithms accordingly. **PTAS from hochbaum shmoys**

Based on various models of describing uncertainty in input parameter, uncertainty problems can be approached by various methodologies including Reactive, stochastic, fuzzy and Robust approach [12]. The bounded uncertainty model assumes that an input parameter have value between a lower and upper bound and is usually dealt with robust approach or sensitivity analysis**cite**. We are more interested in robust approach to deal with uncertainty. **Be factual, this paper takes that approach and shows that or applies it to that.** Wierman and Nuyens [20] introduce SMART as classification to understand size based policies and draw analytical co-relation between response time and estimated job size in single server problem. Dealing with uncertainty problems with robust approach is widely used in practicality in the area of MapReduce [11] [17], Hadoop [21] [19], databases [13] and web servers [3]. HSFS and FLEX schedulers are proposed which provides robustness in scheduling against uncertain Job Size [21] [14]. Cannon and Jeannot [2] provides scheduling heuristics that optimizes both makespan and robustness in scheduling task graph on heterogeneous system. **This is not the contribution of this paper. It is mostly trying to understand what is the correct metric for measuring the robustness of a schedule.**

Most of the work on robust scheduling uses scenarios in problem formulation to structure variability of uncertain parameter. **What does a scenario means in this context.** Daniels and Kouvelis [5] used scenarios to formulate branch and bound based robust scheduling to cope uncertainty in processing time of jobs in single machine. Davenport, Gefflot, and Bek used slack based technique to add extra 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 [8]. Their notion of perturbed processing times is similar to our definition of estimated processing times which can increase or decrease to actual processing times once the jobs are processed. But unlike our work they considered increment and decrement of job processing times as different problem scenario. We have approached the problem with worst case scenario where some tasks may increase and some may decrease in the same schedule.

This is related works, this is not about what we do. We have used pro-active approach to deal with uncertainty. Through this research we study effect of load balancing in scenario of uncertain processing times. Based on different task assignment criteria we propose 3 models for our problem definition, offering different degree of task replication for load balancing. 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][15]. Our notion of using replication is to increase data availability thereby enhancing system reliability against uncertainties. With focus on studying replication affect on performance of a schedule under uncertainty, we have not considered cost of replication in terms of memory utilization. Our work considers an ideal case where storage is sufficient. Similar work is done by Tse [16] who used selective replication of documents in bi criteria problem of minimizing load and memory in distributed file servers.

A. *Our Contribution*

Why do we have contribution here?

We study the effect of load balancing through replication in classic problem of scheduling jobs with uncertain processing times with objective to optimize makespan. We propose 3 models based on different degree of of replication they allow: a)No Replication i.e. $|M_j| = 1$,b) Replication is done every where i.e.$|M_j| = |M|$ and c) Replication in groups i.e $|M_j| = m/k$ where k is number of groups each having m/k processors . For each of the models we propose a two phase algorithm based on Graham's LS and LPT; and derive performance ratio for each. For $|M_j| = 1$, we prove that there is no algorithm which give better performance than α^2. Our algorithm for the model give $\frac{2\alpha^2 m}{2\alpha^2+m-1}$ approximation. For models with replications $|M_j| = |M|$ and $|M_j| = m/k$ the algorithms give performance ratio of $1 + (\frac{m-1}{m})\frac{\alpha^2}{2}$ and $\frac{k\alpha^2}{\alpha^2+k-1}\left[1 + \frac{k-1}{m}\right] + \frac{m-k}{m}$ respectively.

IV. Strategy 1: No Replication

This section model 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: for each task, there is only one machine is can be scheduled 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. He 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 and 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 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, $C_{max} = \alpha B$ and $C^*_{max} \leq \frac{1}{\alpha}\lceil\frac{\lambda m - B}{m}\rceil + \alpha\lceil\frac{B}{m}\rceil$. Figure 1 depicts the online solution and the offline optimal. We have,

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

This proof is incorrect, we need the other inequality! We need a $C^*_{max} \leq something$ so that we will have $Cmax/Cmax* \geq somethingelse$. This should be easy to fix. Is taking the ceil of the expression of $Cmax$ enough?

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 smaller the value of B, 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 $\frac{\alpha^2 m\lambda}{\lambda(\alpha^2+m-1)+m(\alpha^2+1)}$. 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 [10]: The tasks

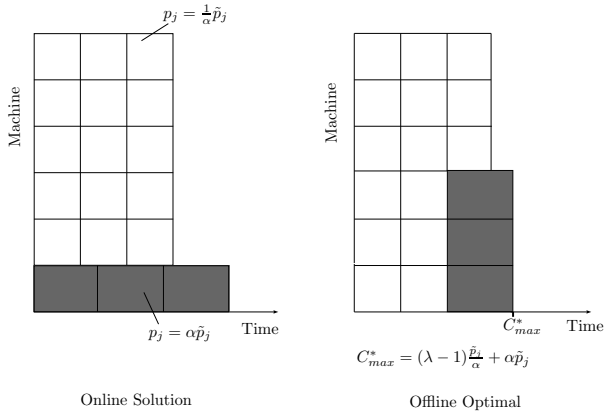


Figure 1: Instance constructed by the adversary in the proof of theorem 1 with $\lambda = 3$ and $m = 6$. In the online solution, the adversary increased by a factor of α the processing time of the task of the most loaded machine. If that information was available beforehand, an optimal offline algorithm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

are sorted in non-increasing order of their processing time and are greedily scheduled on the processor that minimizes the sum of the $\tilde{p_i}$ 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 and on the existence of a better arrangement would the actual processing time be known. The following theorem states the theoretical guarantee of the algorithm.

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 [10], 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 processing that reaches having $\tilde{C}_{max}$, 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 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 times of the task. That is to say, the tasks are sorted in non-increasing order of their estimated processing time. Then the task are scheduled in a 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 in model 3.

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.

$$C^*_{max} \geq \frac{\sum_{i \in J} p_i}{m}$$

$\sum_{i \in T} p_i$ can be written as sum of load on $G1$ and load on rest of groups.

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

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 $max_{i \in J}\tilde{p}_i$. So, for any group $Gl \neq G1$, We have

$$\forall l \in \{2,3,\ldots,k\}, |\sum_{i \in G1} \tilde{p}_i - \sum_{i \in Gl} \tilde{p}_i| \leq max_{i \in T}\tilde{p}_i$$

Adding for all values of l leads to

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

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

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

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

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

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

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,

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

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

From Equation 8 and 7, we derive

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

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

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

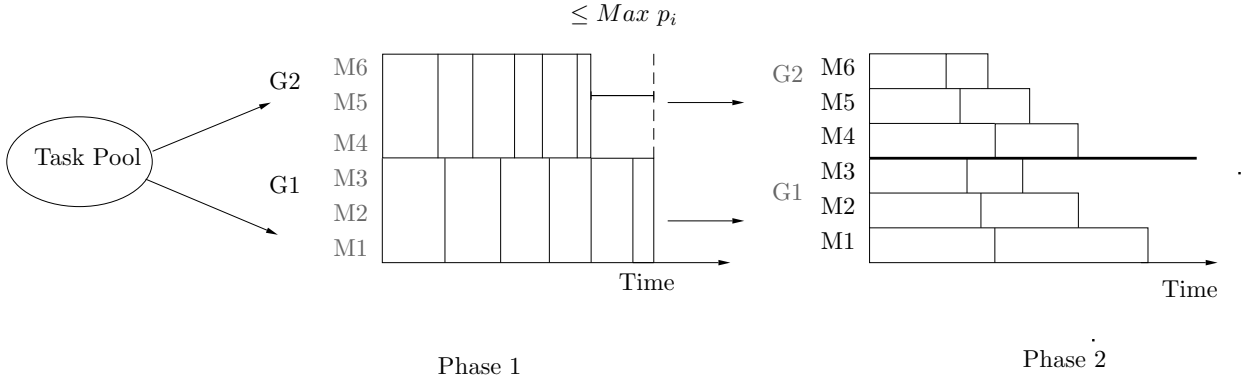


Figure 2: An example of replication in groups with $m = 6$, $k = 2$. In phase 1, the data of the tasks are assigned to one of the groups. Phase 2 schedules each task assigned to a machine within its group.

Using 9 and 10, We have

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

As $C^*_{max} \geq max_{i\in J} p_i \geq p_{max}$, 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 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 $(k-1)\sum_{i\in G1} \tilde{p}_i \leq \sum_{l=2}^{k} \sum_{i\in Gl} \tilde{p}_i$.

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

$$\sum_{l=2}^{k} \sum_{i\in Gl} p_i \geq \frac{1}{\alpha^2}(k-1) \sum_{i\in G1} p_i$$

Putting this value in Equation 7, we have

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

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

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

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

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 α^2 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. 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

Replication	Approximation ratio
$\|M_j\| = 1$	$\frac{C_{max}}{C^*_{max}} \leq \frac{2\alpha^2 m}{2\alpha^2+m-1}$ (Th. 2) No approximation better than $\frac{\alpha^2 m}{\alpha^2+m-1}$ (Th. 1)
$\|M_j\| = m$	$\frac{C_{max}}{C^*_{max}} \leq 1 + (\frac{m-1}{m})\frac{\alpha^2}{2}$ (Th. 4) $\frac{C_{max}}{C^*_{max}} \leq 2 - \frac{1}{m}$ [9]
$\|M_j\| = \frac{m}{k}$	$\frac{C_{max}}{C^*_{max}} \leq \frac{k\alpha^2}{\alpha^2+k-1}\left(1 + \frac{k-1}{m}\right) + \frac{m-k}{m}$ (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.

the number of machines fixed $m = 210$.

When $\alpha = 1.1$, there is **LS-Groups** 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-Groups** with only 1 group.

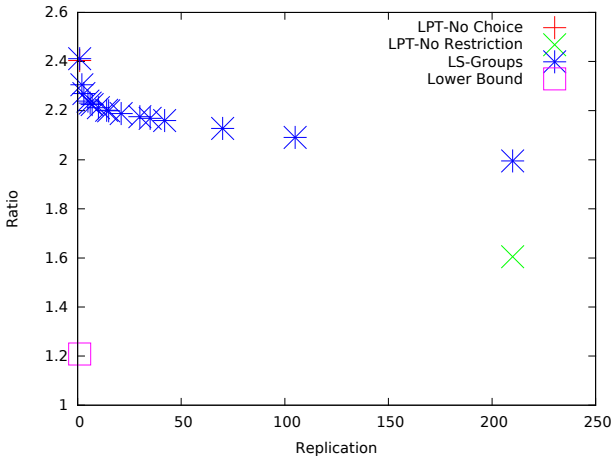
When α increases to 1.5, there is no more differences in the guarantees of **LS-Groups** with 1 group and **LPT-No Restriction**. Also **LS-Groups** provides many intermediate solution between deploying the data on a single machine and deploying them everywhere.

When $\alpha = 2$, the range of the approximation ratios increase and the value of the lower bound increases. Now **LS-Groups** 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.

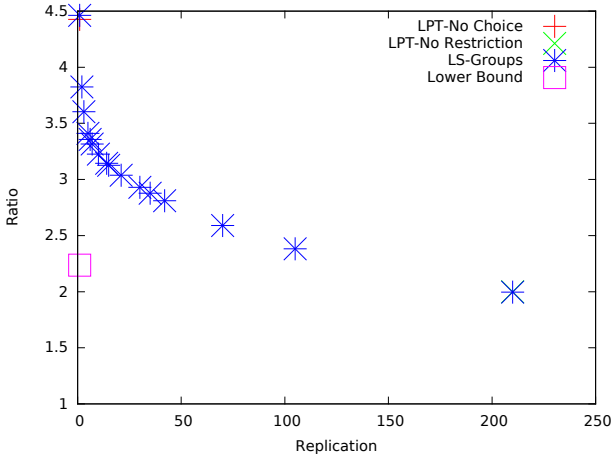
Overall, when α is large, only few replication improve the performance significantly.

VIII. Conclusion and Future Work

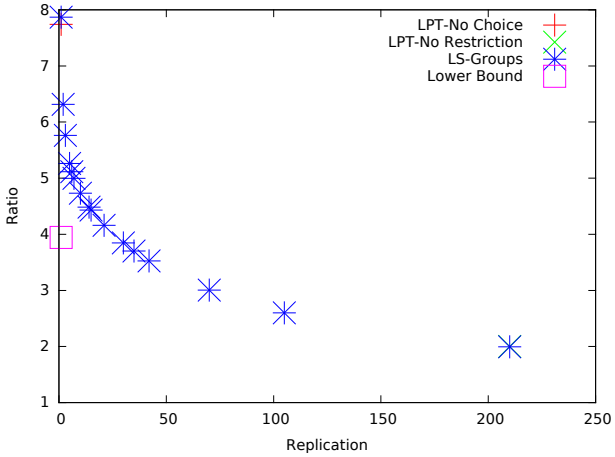
We studied the effect on uncertainty in the processing time of tasks on scheduling for parallel and distributed machines. In particular, we investigated how allowing tasks to execute on different machines can help dealing with not knowing the processing time of tasks accurately. We investigated three 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 particular, we obtained a better guarantee with few 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



(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 $\alpha \in \{1.1, 1.5, 2\}$.

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. Springer Berlin Heidelberg, 2004.

[2] L.-C. Canon and E. Jeannot. Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments. 2010. Accepted for publication.

[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.

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

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

[10] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM JOURNAL ON APPLIED MATHEMATICS*, 17(2):416–429, 1969.

[11] 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, Washington, DC, USA, 2010. IEEE Computer Society.

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

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

[14] 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.

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

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

[17] 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.

[18] D. Wang, G. Joshi, and G. W. Wornell. Efficient task replication for fast response times in parallel computation. *CoRR*, abs/1404.1328, 2014.

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

[20] A. Wierman and M. Nuyens. Scheduling despite inexact job-size information. In Z. Liu, V. Misra, and P. J. Shenoy, editors, *SIGMETRICS*, pages 25–36. ACM, 2008.

[21] 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, Berlin, Heidelberg, 2010. Springer-Verlag.

[22] 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.