DATA REPLICATION FOR COPING WITH UNCERTAINTY IN SCHEDULING

by

Manmohan Chaubey

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computing and Information Systems

Charlotte

2014

Approved by:

____

Dr. Erik Saule

____

Dr. Yu Wang

____

Dr. Aidong Lu

ABSTRACT

MANMOHAN CHAUBEY. Data Replication for coping with Uncertainty in Scheduling. (Under the direction of DR. ERIK SAULE)

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.

This thesis research investigates how replicating the data required by the tasks can help coping with the inaccuracies of the processing times. In particular, it investigates 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, this thesis investigates three different strategies, each allowing a different degree of replication of jobs: a) No Replication b) Replication everywhere and c) Replication in groups, and proposes 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. Replication improves performance but incurs a cost in terms of memory consumption. The objective is then to develop scheduling algorithm with good competitive ratio to minimize both the makespan of the schedule and the memory consumption of the machines.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

This chapter provides the foundations for the principle objective of this dissertation, which is to investigate the effect of task replication on scheduling under uncertainty of processing times. The subject matter of this thesis falls in the intersection of several areas of current research interest. These includes: (1) Scheduling under uncertainty, (2) Data placement and Replication strategies to improve performance of a schedule, and (3) Bi-objective optimization for simultaneously optimizing makespan as well as memory consumption

1.1 Motivation

In real world scheduling problems often the parameters such as processing time of a task is not known exactly in advance.The goal of an scheduling algorithm is to generate robust schedule against uncertainty. Dealing with uncertainty is difficult as in real world problems a task can be processed on particular computing systems, other wise a task could be moved as system sees it fit without incurring extra cost and the problem would be vastly alleviated. But in practice a task has to run on particular machine especially in " out of core" computing applications which involve very large data sets. For example, solving systems of linear equations and computing eigenvalues – where matrices involved are very large. When the data sets are too large to fit in the main memory of a computer, it must be stored on any external memory source such as disks. Disk storage is significantly cheaper than main memory storage. However, accessing data from a disk is relatively slower than accessing the main memory. So, a scheduling algorithm in " out of core" execution places data to main memory of different systems such that data access from disk or any external storage is reduced. That means a task is pinned to a particular computing unit. So handling uncertainty in out of core execution is having added overhead. Hence, developing a scheduling strategy which can guarantee performance under uncertainty of processing times of the tasks with restriction that a task can be scheduled to particular set of machines motivates this research.

Scheduling tasks on distributed memory is prevalent in Hadoop. Hadoop-MapReduce constitute a powerful Computation Model for processing large data sets on distributed clusters [21] . Hadoop uses Hadoop Distributed File System (HDFS) to store large amount of data across multiple machines. MapReduce provides a framework for processing the data across multiple machines. Task execution time in such a large scale is not a precise time but naturally contains uncertainties in terms of node failure and tasks failure. To provide robustness against uncertainty Hadoop uses data replication across multiple nodes. A good memory aware replication strategy balances the load across multiple machines and can provide reliability against uncertainty. One of the main goal of a Hadoop system is to maintain node locality which means running data on the node that contains it [32]. Therefore, a data intensive scheduling incorporating data location and choosing popular data sets to replicate would be beneficial [13]; and serves to provide motivation for this research.

1.2 Scheduling

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 [18], because prediction is hard in the general case [29], or because underlying models of a particular algorithm can only predict runtime within a given range [8]. Whichever the reason is, not knowing accurately the processing time can significantly impact the performance obtained from the machine.

For instance in out-of-core sparse linear algebra, executing a task where the data are not locally available would have a prohibitive overhead [34, 33].

One approach for dealing with the uncertainty of processing time is to build a robust schedule [2, 10**?**], 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 thesis pursues the idea of replicating 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 [27]. And it has been shown that launching the same task multiple times can help cope with hardware differences [26] 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 [33, 34]). This research answer the question "can data replication help cope with the uncertainty of processing time?" And the answer is that it can.

1.3 Research Contribution

This thesis proposes strategies and presents algorithms to cope with uncertainty in processing times of the tasks. The research provides three replication strategies and studies the tradeoff between the number of replication and the guarantee on the makespan. The strategy **No Replication** 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. The strategy **Replicate data everywhere** 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. The strategy **Replication in groups** 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.

To alleviate the cost of replication in terms of memory consumption the thesis presents two memory-aware algorithms to optimize the makespan as well as the memory consumption. The proposed algorithms divides the tasks into two sets: memory intensive tasks and processing time intensive tasks and schedule differently to minimize both the objectives.

1.4 Thesis Outline

The remaining of this thesis is organized as follows: we describe system model and notations in Chapter 2. Related works are presented in Chapter 3. Chapter 4 investigates the effect of replication on processing time uncertainty through three strategies which offer different degree of replication of the tasks. The chapter summarizes the various results derived for each strategy and studies the tradeoff between performance guarantee and data replication. Chapter 5 investigates bi-objective problem of minimizing the makespan as well as the memory usage and proposes two memory-aware algorithms which simultaneously optimizes both the objectives. Chapter 6 concludes the thesis with remarks and raises few challenging questions which could be future research topics.

CHAPTER 2: 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. Each task j occupies s_j space in memory. 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:

$$\frac{\tilde{p_j}}{\alpha} \leq p_j \leq \alpha\tilde{p_j} \tag{1}$$

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 [31] 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 [8] 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 parallel system scheduling can be modeled into different objective functions with different parameters to optimize. A makespan minimization problem has objective to minimize completion time of last task of the system. Memory is another parameter for objective function. A memory aware scheduling aims at minimizing total memory consumption $\sum_j s_j$ or memory consumption of most occupied machine $max_i \sum_{j \in E_i} s_i$. Replication improves processing of tasks but increases memory consumption in the system. So, objective function attached with replication can be where to replicate tasks and which tasks to replicate so that performance can improve without violating any memory constraint or with bi-objective to minimize memory also along with improving processing time of the tasks.

In Chapter 4, 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). The memory objective is constrained by allowing different degree of replication by choosing where (on which set of machines) a task to be replicated. 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.

In Chapter 5, we tackle the bi-objective problem of simultaneously minimizing makespan C_{max} as well as memory usage, $M_{max} = \max_i \sum_{j \in E_i} s_j$ which denotes the maximum memory usage of a machine. As a task occupies fixed amount of memory but its processing time is uncertain, both objectives are asymmetrical. M^*_{max} denotes optimal maximum memory consumption of a machine. An algorithm generates a schedule which is ρ^C-approximated on makespan and ρ^M-approximated on memory.

There are two ways to deal with multi objective optimization [23] [7]:

1. Epsilon-constraint method: This approach optimizes the primary objective setting the other objective within some constraint . We use this approach in chapter 4 to optimize makespan setting the memory objective by allowing different degree of replication of the tasks.

2. Zenith approximation: This approach optimizes both the objective at the same time. We use this approach to optimize both makespan and memory usage in chapter 5.

CHAPTER 3: RELATED WORK

This chapter provides the literature review on related research areas such as uncertainty in scheduling, data placement and replication. For better understanding the core concept of this thesis research proofs of some classical scheduling algorithms is presented along with a brief introduction in the context they appear while literature review.

3.1 Classical Scheduling problem

When $\alpha = 1$, the problem is exactly the classical independent tasks scheduling problem on identical machines, which is known to be NP-Hard [9]. We use Graham's List Scheduling (LS) [11] and Largest Processing Time (LPT) algorithms [12] 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 [14].

We begin with by recalling the formal proofs of the guarantees of LS and LPT algorithms:

Property 1. *[11] List Scheduling has an approximation ratio of* $2 - \frac{1}{m}$.

Proof. Let l be the last task in the system which is processed on machine r and it starts on r at time t. Clearly, the makespan C_{max} of the schedule is $t + p_l$. As in LS a new task is scheduled on the least loaded machine at that time, for each machine i, we have $t \leq \sum\limits_{j \in E_i} p_j$. Adding this for all the machines including r, we get $mt \leq \sum\limits_{i \in M-\{r\}} \sum\limits_{j \in E_i} p_j + \sum\limits_{j \in E_r} p_j - p_l \Rightarrow t \leq (\sum\limits_{j} p_j - p_l)/m$. Hence, $C_{max} \leq \frac{\sum_j p_j + (m-1)p_l}{m}$.

The optimal makespan of a schedule C^*_{max} must be greater or equal to the average load over all the m machines, $C^*_{max} \geq \frac{\sum_j p_j}{m}$. Also, C^*_{max} cannot be smaller than any task in the system, hence $C^*_{max} \geq Maxp_j \geq p_l$. Therefore, $C_{max} \leq C^*_{max} + C^*_{max}(m - 1)/m$. Hence, $C_{max}/C^*_{max} \leq 2 - 1/m$. □

Property 2. *[12] The LPT algorithm has an approximation ratio of* $\frac{4}{3} - \frac{1}{3m}$.

Proof. LPT always generates an optimal schedule if no machine has more than 2 tasks. So to derive an approximation ratio we can assume that there are at least 3 tasks in a machine. As LPT assigns tasks to machines in non-increasing order of their processing times, the last task l is the smallest task in the machine. Since there are at least 3 tasks in a machine $C^*_{max} \geq 3p_l$. Also, $C^*_{max} \geq \frac{\sum_j p_j}{m}$ and $C_{max} \leq \frac{\sum_j p_j + (m-1)p_l}{m}$ as shown in previous proof. Therefore, $C_{max} \leq C^*_{max} + C^*_{max}(m - 1)/3m$. Hence, $C_{max}/C^*_{max} \leq 4/3 - 1/3m$. □

3.2 Uncertainty and Robustness

Based on various models for describing the uncertain input parameter, various methodologies can be used including reactive, stochastic, fuzzy and robust approach [16].

We are using the bounded uncertainty model which assumes that an input parameter have value between a lower and upper bound. Wierman and Nuyens [28] introduce SMART, a classification to understand size-based policies and draw analytic co-relation between response time and estimated job size in single server problem. Robust approaches to deal with uncertainty are widely used on MapReduce systems [15] [25], in Hadoop [30] [27], on databases [17] and on web servers [3]. The HSFS and FLEX schedulers provide robustness in scheduling against uncertain job size [30, 19]. 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 [10]. 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.

3.3 Data placement and Replication

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][20]. Tse [24] 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. Our approach for bi- criteria optimization of makespan and memory consumption is based on the $SBO_\triangle$ Algorithm proposed by Saule, Dutot and Mounie [22]. The algorithm uses ρ_1 and ρ_2 approximated independent schedules on makespan and memory consumption respectively, and it computes a $((1+\triangle)\rho_1, (1+\frac{1}{\triangle})\rho_2)$- approximated schedule with $\triangle$ as a parameter of the algorithm.

Property 3. *[22] The $SBO_\triangle$ Algorithm generates a $(1+\triangle)\rho_1$-approximated schedule on makespan.*

Proof. The algorithm schedules a task j according to a π_2 schedule generated by the ρ_2-approximated algorithm on memory if it satisfies this condition: $\frac{p_j}{C_{max}^{\pi_1}} \leq \triangle \frac{s_j}{M_{max}^{\pi_2}}$. Where $C_{max}^{\pi_1}$ is the makespan obtained using a π_1 schedule generated by the ρ_1-approximated algorithm on makespan, and $M_{max}^{\pi_1}$ is the memory consumption of the most occupied machine obtained using π_2. If this condition is not satisfied the task is scheduled according π_1. Let k be the machine reaching makespan C_{max} of the schedule generated by the $SBO_\triangle$ algorithm. Let S_1 be the set of tasks scheduled according π_1 and S_2 be the set of tasks scheduled according π_2 schedule. C_{max} can be decomposed as the sum of the processing times of the tasks in set S_1 and S_2 scheduled on machine k.

$$C_{max} = \sum_{j \in S_1 \cap E_k} p_j + \sum_{j \in S_2 \cap E_k} p_j$$

Since $C^{\pi_1}_{max} \geq \sum_{j \in S_1 \cap E_k} p_j$ and $\sum_{j \in S_2 \cap E_k} \triangle C^{\pi_1}_{max} \frac{s_j}{M^{\pi_2}_{max}} \geq \sum_{j \in S_2 \cap E_k} p_j$ by definition of S_2, we have

$$C_{max} \leq C^{\pi_1}_{max} + \sum_{j \in S_2 \cap E_k} \triangle C^{\pi_1}_{max} \frac{s_j}{M^{\pi_2}_{max}}$$

Since $\sum_{j \in S2 \cap E_k} \frac{s_j}{M^{\pi_2}_{max}} \leq 1$, we have

$$C_{max} \leq (1 + \triangle) C^{\pi_1}_{max}$$

Since $C^{\pi_1}_{max} \leq \rho_1 C^*_{max}$, the algorithm has an approximation ratio of $(1 + \triangle)\rho_1$ on the makespan. □

Property 4. *[22] The SBO$_\triangle$ Algorithm generates a $(1+\frac{1}{\triangle})\rho_2$-approximated schedule on memory.*

Proof. Let k be the machine with most memory consumption. Similar to the previous proof, M_{max} can be written as the sum of memory usage of the tasks in sets S_1 and S_2 scheduled on machine k, $\sum_{j \in S_1 \cap E_k} s_j + \sum_{j \in s_2 \cap E_k} p_j$. Since, $\sum_{j \in S_2 \cap E_k} s_j \leq M^{\pi_1}_{max}$ and by definition of S_1, $\sum_{j \in S_1 \cap E_k} s_j \leq \frac{M^{\pi_1}_{max}}{\triangle C^{\pi_2}_{max}} \sum_{j \in S_2 \cap E_k} p_j \leq \frac{M^{\pi_1}_{max}}{\triangle}$, we have

$$M_{max} \leq (1 + \frac{1}{\triangle}) M^{\pi_1}_{max}$$

Since $M^{\pi_1}_{max} \leq \rho_2 M^*_{max}$, the algorithm has an approximation ratio of $(1 + \frac{1}{\triangle})\rho_2$ on memory. □

CHAPTER 4: REPLICATED DATA PLACEMENT STRATEGIES

This chapter provides three strategies, each offering different degree of replication to study the tradeoff between the number of replication and the guarantee on the makespan. The strategy **No Replication** restricts that each task can be scheduled to only one machine and allows no replication of the tasks. The strategy **Replicate data everywhere** replicates data everywhere and studies what can be achieved in doing so. The strategy **Replication in groups** replicates data in group of processors and act an intermediate strategy between the previous two strategies.

4.1 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.

Theorem 5. *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

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

□

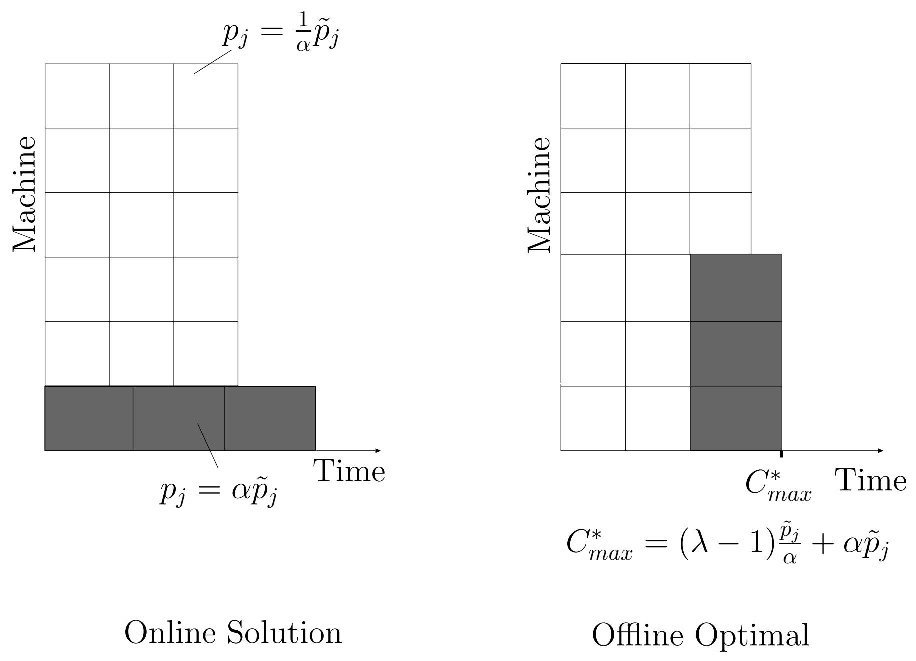

$p_j = \frac{1}{\alpha}\tilde{p}_j$
Machine
Time
$p_j = \alpha\tilde{p}_j$
Online Solution
Machine
C^*_{max}
Time
$C^*_{max} = (\lambda - 1)\frac{\tilde{p}_j}{\alpha} + \alpha\tilde{p}_j$
Offline Optimal

Figure 1: Instance constructed by the adversary in the proof of theorem 5 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.

Corollary 5.1. *When m goes to ∞ there is no online algorithm having competitive ratio better than α^2.*

4.1.1 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 [12]: 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.

Theorem 6. *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 [12], 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,

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

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

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

□

4.2 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 7. *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 8. ***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 7, 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 8. Since **LPT-No Restriction** is a variant of List Scheduling, the algorithm has a competitive ratio of $\min(1 + \frac{m-1}{2m}\alpha^2, 2 - \frac{1}{m})$.

4.3 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.

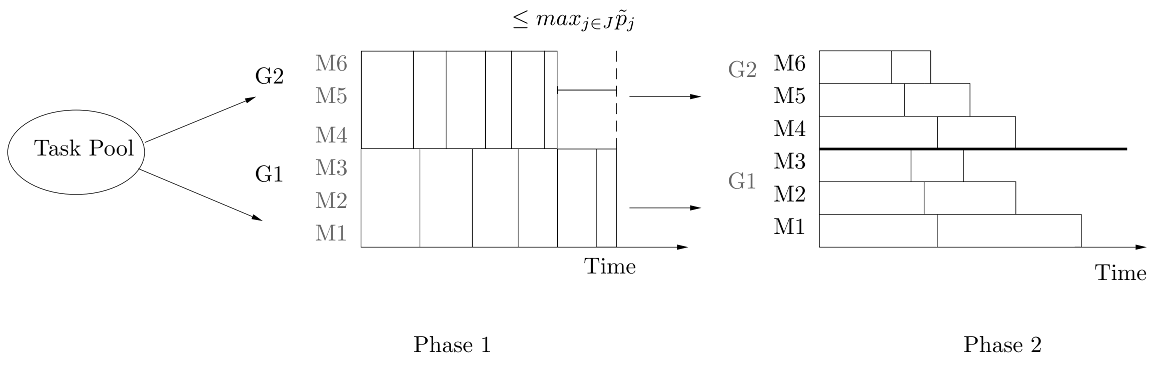


Figure 2: An example of replication in groups with $m = 6$, $k = 2$. In phase 1, the data of the tasks are assigned to one of the groups. Phase 2 schedules each task assigned to a machine within its group.

Theorem 9. *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.

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

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

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

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

Putting this value in Equation 7, we have

$$C^*_{max} \geq \frac{\alpha^2+k-1}{m\alpha^2}\sum_{j\in G1} p_j \tag{11}$$

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.

4.4 Summary

Table 1 summarizes the results of this chapter in term of approximation theory. Based on adversary technique, Theorem 5 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 $\frac{k\alpha^2}{\alpha^2+k-1}\left(1 + \frac{k-1}{m}\right) + \frac{m-k}{m}$.

Replication	Approximation ratio
$\|M_j\| = 1$	$\frac{C_{max}}{C^*_{max}} \leq \frac{2\alpha^2 m}{2\alpha^2+m-1}$ (Th. 6) No approximation better than $\frac{\alpha^2 m}{\alpha^2+m-1}$ (Th. 5)
$\|M_j\| = m$	$\frac{C_{max}}{C^*_{max}} \leq 1 + (\frac{m-1}{m})\frac{\alpha^2}{2}$ (Th. 8) $\frac{C_{max}}{C^*_{max}} \leq 2 - \frac{1}{m}$ [11]
$\|M_j\| = \frac{m}{k}$	$\frac{C_{max}}{C^*_{max}} \leq \frac{k\alpha^2}{\alpha^2+k-1}\left(1 + \frac{k-1}{m}\right) + \frac{m-k}{m}$ (Th. 9)

Table 1: Summary of the contribution of this chapter. Three proposed algorithms have guaranteed performance. One lower bound on approximability has been established.

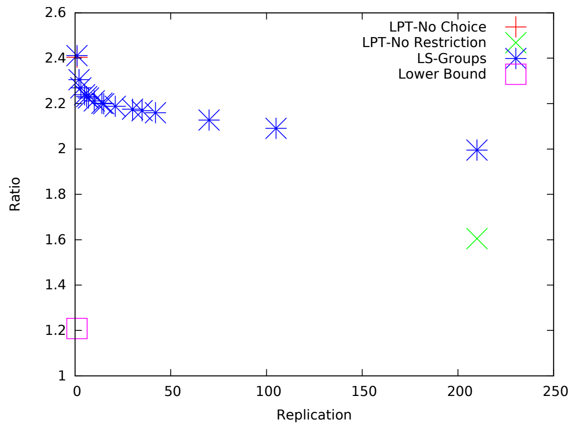
Of course, there is an inherent tradeoff between replicating data and obtaining good values for the makespan. To better understand the tradeoff we show in Figure 3 how the expressions of the guarantees (or impossibility)translate to actual values in a approximation ratio / replication space. We picked 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.

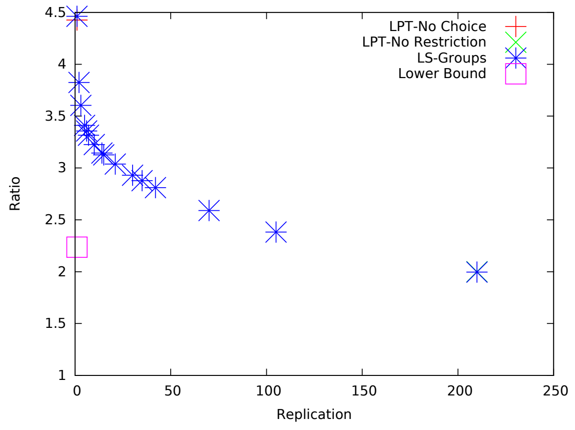
When α increases to 1.5, there is no more differences in the guarantees of **LS-Group** with 1 group and **LPT-No Restriction**. Also **LS-Group** provides many intermediate solution between deploying the data on a single machine and deploying them everywhere.

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.

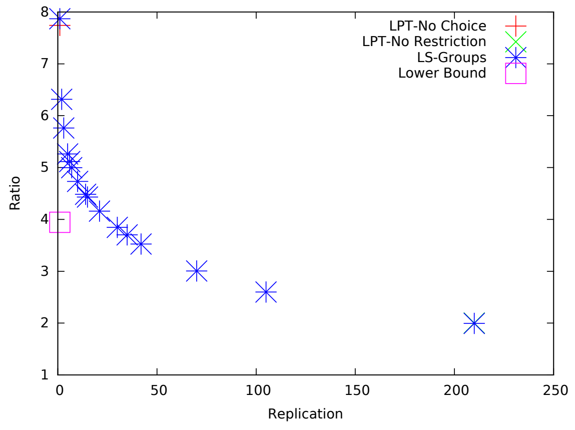
Overall, when α is large, only few replication improve the performance significantly.



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



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



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

Figure 3: Ratio-Replication graph with $m = 210$ and $\alpha \in \{1.1, 1.5, 2\}$.

CHAPTER 5: MEMORY AWARE REPLICATION UNDER UNCERTAINTY

Replication improves performance but incurs a cost in terms of memory consumption. Replication allows to obtain a better load balancing by reducing the effect of uncertainties in processing times of tasks. But each replica occupies memory, and increases the memory consumption. So, replicating all the tasks is not possible in real scenarios. This justify the need for an efficient replication strategy which allows an algorithm to choose which tasks are to be replicated and where. In this chapter we investigate the bi-objective problem of minimizing the makespan as well as the memory consumption. A memory-aware replication strategy improves execution times with little increase in memory consumption.

5.1 Preliminaries

The problem is to schedule a set J of n tasks on m machines such that both makespan C_{max} as well as memory usage M_{max} is optimized. Let π_1 be the schedule which minimizes makespan and π_2 be the memory-aware schedule. $\tilde{C}^{\pi_1}_{max}$ and $\tilde{C}^{*}_{max}$ are makespan and optimal makespan when all the tasks are scheduled according to π_1. Similarly, $M^{\pi_2}_{max}$ is memory consumption of the most occupied machine and M^{*}_{max} is its optimal value. The strategy is to divide tasks into two sets S_1 and S_2 such that set S_1 contains the processing time intensive tasks and set S_2 contains the memory intensive tasks, and schedule them differently and in such a way that it optimizes both the objectives.

We propose two algorithms $SABO_{\triangle}$ (stands for static asymmetric bi-objective) and $ABO_{\triangle}$ (stands for asymmetric bi-objective), which are based on $SBO_{\triangle}$ algorithm. $SBO_{\triangle}$ [22] is bi-objective algorithm for minimizing makespan and memory usage for independent tasks by combining results of two symmetric schedules each dedicated to a single objective.

5.2 The $SABO_{\triangle}$ Algorithm

We propose $SABO_{\triangle}$ Algorithm which is static in nature and restrict each task to be scheduled to only one machine. Similar to $SBO_{\triangle}$ this algorithm assigns tasks to all the machines in phase 1 such that it minimizes both the objectives. As each task is restricted to only one machine, there is no task replication. Based on similar condition as in $SBO_{\triangle}$, a processing-time intensive task is assigned to π_1 schedule and a memory intensive task is assigned to π_2

In phase 2, the algorithm loads the tasks to the machines they were assigned in phase 1.

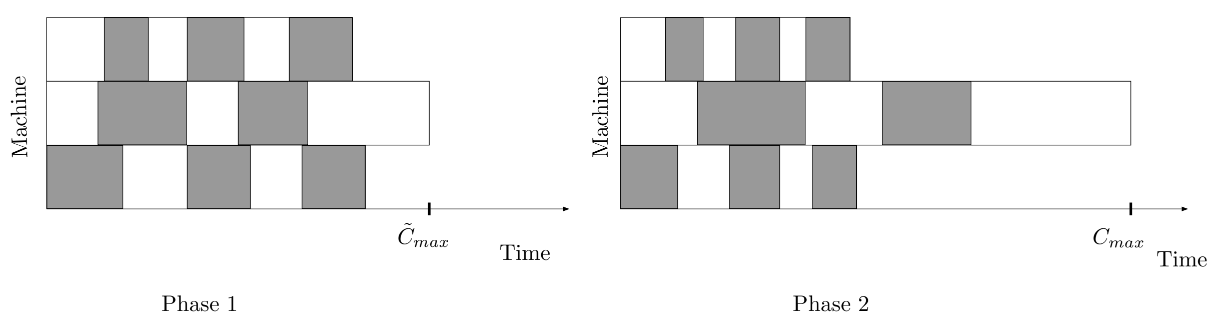


Figure 4: An example of two phases of the schedule generated by the $SABO_{\triangle}$. The uncolored parts represent tasks scheduled according π_2. The colored parts represents tasks scheduled according π_1

Algorithm 1 $SABO_{\triangle}$

```
Input: $m$ machines
        Set $J$ of $n$ tasks
        Let $\pi_1$ be a $\rho_1$-approximated schedule on makespan $\tilde{C}_{max}$
        Let $\pi_1$ be a $\rho_2$- approximated schedule on memory $M_{max}$

Phase 1: [Uses $SBO_{\triangle}$]
for all $j \in S_1$ do
    if $\frac{\tilde{p}_j}{\tilde{C}^{\pi_1}_{max}} \leq \triangle \frac{s_j}{M^{\pi_2}_{max}}$ then
        Assign $j$ to a machine according to $\pi_2$ schedule
        Add $j$ to $S_2$
    else
        Assign $j$ to a machine according to $\pi_1$ schedule
        Add $j$ to $S_1$
    end if
end for
End of Phase 1

Phase 2:
        Schedule tasks to machines to which they were assigned during phase 1
End of Phase 2
```

Theorem 10. *The $SABO_{\triangle}$ Algorithm generates a $(1+\triangle)\alpha^2\rho_1$ - approximated schedule on makespan.*

Proof. Let k be the machine reaching the makespan C_{max} of the schedule. C_{max} can be written as the sum of processing times of tasks in set S_1 and S_2 scheduled on machine k.

$$C_{max} = \sum_{j \in S_1 \cap E_k} p_j + \sum_{j \in S_2 \cap E_k} p_j$$

Since, $\sum_{j \in S_2 \cap E_k} p_j \leq \alpha \sum_{j \in S_2 \cap E_k} \tilde{p}_j$

$$C_{max} \leq \sum_{j \in S_1 \cap E_k} p_j + \alpha \sum_{j \in S_2 \cap E_k} \tilde{p}_j$$

Let $C^{\pi_1}_{max}$ denotes the makespan obtained after phase 2 when tasks are loaded and actual processing time of a task is known to scheduler. Since $C^{\pi_1}_{max} \geq \sum_{j \in S_1 \cap E_k} p_j$ and

$\sum\limits_{j \in S_2 \cap E_k} \triangle \tilde{C}^{\pi_1}_{max} \frac{s_j}{M^{\pi_2}_{max}} \geq \sum\limits_{j \in S_2 \cap E_k} \tilde{p}_j$ by definition of S_2, we have

$$C_{max} \leq C^{\pi_1}_{max} + \alpha \sum_{j \in S_2 \cap E_k} \triangle \tilde{C}^{\pi_1}_{max} \frac{s_j}{M^{\pi_2}_{max}}$$

Since, $C^{\pi_1}_{max} \leq \alpha \tilde{C}^{\pi_1}_{max}$ and $\sum\limits_{j \in S_2 \cap E_k} \frac{s_j}{M^{\pi_2}_{max}} \leq 1$, we have

$$C_{max} \leq (1 + \triangle)\alpha \tilde{C}^{\pi_1}_{max}$$

Since $\tilde{C}^{\pi_1}_{max} \leq \rho_1 \tilde{C}^{*}_{max} \leq \alpha \rho_1 C^{*}_{max}$ the algorithm has an approximation ratio of $(1 + \triangle)\alpha^2 \rho_1$ on makespan. □

Theorem 11. *The $SABO_\triangle$ Algorithm generates $(1 + \frac{1}{\triangle})\rho_2$- approximated schedule on memory*

Proof. The proof is identical to $SBO_\triangle$ algorithm and is presented in chapter 3. □

5.3 The $ABO_\triangle$ Algorithm

We propose a two phase algorithm. In phase 1 the algorithm assigns tasks to all the machines such that it minimizes both the makespan as well as memory consumption. The tasks having more memory value in comparison to its processing time are scheduled using memory intensive schedule which aim at minimizing memory. Similarly tasks which incur more processing time cost compared to memory cost are assigned to machines according to the makespan intensive schedule. These tasks are replicated to all machines in order to provide better load balancing and hence minimized makespan. The algorithm in its phase 1 assigns all the memory intensive tasks to machines first, then chooses tasks having more processing time values compared to memory they consume.

In phase 2, the algorithm loads the memory intensive tasks to the machines they were assigned in phase 1 respecting the tasks assignment during phase 1. The algorithm schedule the time intensive tasks (replicated tasks) using Graham's List Scheduling after all the memory intensive tasks are scheduled. Figure 5 shows a schedule instance using the algorithm.

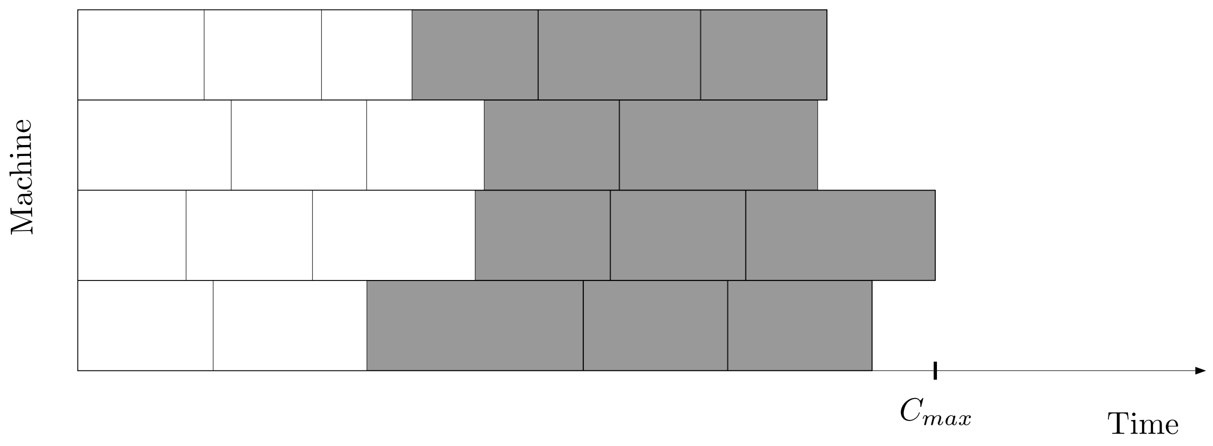


Figure 5: An example of the schedule generated by the $ABO_{\triangle}$ algorithm. The uncolored parts represent the memory intensive tasks scheduled according π_2. The colored parts represent the processing time intensive tasks and scheduled using LS after replicated

Theorem 12. *The $ABO_{\triangle}$ Algorithm generates a $(2 - \frac{1}{m} + \triangle\alpha^2\rho_1)$- approximated schedule on makespan.*

Proof. Let k be the machine reaching the makespan C_{max} of the schedule. C_{max} can be written as the sum of the processing times of tasks in sets S_1 and S_2 scheduled on machine k.

$$C_{max} = \sum_{j \in S_1 \cap E_k} p_j + \sum_{j \in S_2 \cap E_k} p_j$$

Algorithm 2 $ABO_{\triangle}$

```
Input: $m$ machines
       Set $J$ of $n$ tasks
       Let $\pi_1$ be a $\rho_1$-approximated schedule on makespan $\tilde{C}_{max}$
       Let $\pi_2$ be a $\rho_2$- approximated schedule on memory $M_{max}$

Phase 1:
for all $j \in J$ do
    if $\frac{\tilde{p_j}}{\tilde{C}_{max}^{\pi_1}} \leq \triangle \frac{s_j}{M_{max}^{\pi_2}}$ then
        Assign $j$ to a machine according to $\pi_2$ schedule
        Add task $j$ to set $S_2$
    end if
end for
for all $j \in S_1$ do
    if $\frac{\tilde{p_j}}{\tilde{C}_{max}^{\pi_1}} \geq \triangle \frac{s_j}{M_{max}^{\pi_2}}$ then
        Add $j$ to set $S_1$
        Replicate $j$ everywhere
    end if
end for
End of Phase 1

Phase 2:
       Schedule tasks from set $S_2$ respecting job assignment during phase 1
       Schedule all replicated tasks from set $S_1$ using Graham's LS Algorithm
End of Phase 2
```

Since, $\sum_{j \in S_2 \cap E_k} p_j \leq \alpha \sum_{j \in S_2 \cap E_k} \tilde{p}_j$

$$C_{max} \leq \sum_{j \in S_1 \cap E_k} p_j + \alpha \sum_{j \in S_2 \cap E_k} \tilde{p}_j$$

Let C^R_{max} denotes makespan obtained by scheduling the replicated tasks using LS. Since $C^R_{max} \geq \sum_{j \in S_1 \cap E_k} p_j$ and $\sum_{j \in S_2 \cap E_k} \triangle \tilde{C}^{\pi_1}_{max} \frac{s_j}{M^{\pi_2}_{max}} \geq \sum_{j \in S_2 \cap E_k} \tilde{p}_j$ by definition of S_2, we have

$$C_{max} \leq C^R_{max} + \alpha \sum_{j \in S_2 \cap E_k} \triangle \tilde{C}^{\pi_1}_{max} \frac{s_j}{M^{\pi_2}_{max}}$$

Using the property of LS, the approximation ratio of the schedule incorporating only replicated tasks is $2 - \frac{1}{m}$. So, $C^R_{max} \leq (2 - \frac{1}{m}) C^*_{max}$. Also, $\sum_{j \in S_2 \cap E_k} \frac{s_j}{M^{\pi_2}_{max}} \leq 1$.

$$C_{max} \leq (2 - \frac{1}{m}) C^*_{max} + \alpha \triangle \tilde{C}^{\pi_1}_{max}$$

Also, $\tilde{C}^{\pi_1}_{max} \leq \rho_1 \tilde{C}^*_{max}$. Since $\tilde{C}^*_{max}$ is the optimal makespan obtained after phase 1 considering estimated processing times of the tasks, we have, $\tilde{C}^*_{max} \leq \alpha C^*_{max}$. So, $\tilde{C}^{\pi_1}_{max} \leq \alpha \rho_1 C^*_{max}$. Using this, we have

$$C_{max} \leq (2 - \frac{1}{m}) C^*_{max} + \alpha^2 \triangle \rho_1 C^*_{max}$$

Hence, we proved that the algorithm generates a $(2 - \frac{1}{m} + \triangle \alpha^2 \rho_1)$- approximated schedule on makespan. $\square$

Theorem 13. *The $ABO_{\triangle}$ Algorithm generates a $(1 + \frac{m}{\triangle})\rho_2$- approximated schedule on memory.*

Proof. When a task is replicated all its replica occupies space in memory and increase memory consumption. For m replicas the total memory consumption is m times of the replicated tasks. Similar to proof of previous theorem, the highest maximum memory occupied by any machine k can be written as

$$M_{max} = \sum_{j \in S_1 \cap E_k} s_j + \sum_{j \in S_2 \cap E_k} s_j$$

As each task in set S_1 is replicated over all the machines, $\sum\limits_{j \in S_1 \cap E_k} s_j = \sum\limits_{j \in S_1} s_j$.

$$M_{max} = \sum_{j \in S_1} s_j + \sum_{j \in S_2 \cap E_k} s_j$$

$\sum\limits_{j \in S_2 \cap E_k} s_j$ at most be equal to $M^{\pi_2}_{max}$ and $\sum\limits_{j \in S_1} s_j$ is bounded by $\sum\limits_{j \in J} M^{\pi_2}_{max} \frac{\tilde{p_j}}{\triangle \tilde{C}^{\pi_1}_{max}}$ as per condition for π_1 scheduling, using this we have

$$M_{max} \leq \sum_{j \in J} M^{\pi_2}_{max} \frac{\tilde{p_j}}{\triangle \tilde{C}^{\pi_1}_{max}} + M^{\pi_2}_{max}$$

Since $\sum\limits_{j \in J} \tilde{p}_j \leq m\tilde{C}^{\pi_1}_{max}$, we have

$$M_{max} \leq \frac{m}{\triangle} M^{\pi_2}_{max} + M^{\pi_2}_{max}$$

Also, $M^{\pi_2}_{max} \leq \rho_2 M^{*}_{max}$. Hence, The Algorithm generate $(1 + \frac{m}{\triangle})\rho_2$- approximated schedule on memory. $\square$

5.4 Summary

Table 2 summarizes the results for $SABO_\triangle$ and $ABO_\triangle$ algorithms. $SABO_\triangle$ is similar to $SBO_\triangle$ algorithm in its first phase and has a approximation ratio of $[(1 + \triangle)\alpha^2\rho_1, (1+\frac{1}{\triangle})\rho_2]$ on makespan and memory. $ABO_\triangle$ is a $[(2-\frac{1}{m}+\triangle\alpha^2\rho_1), (1+\frac{m}{\triangle})\rho_2]$-approximated algorithm on makespan and memory and replicate processing time intensive tasks to improve makespan.

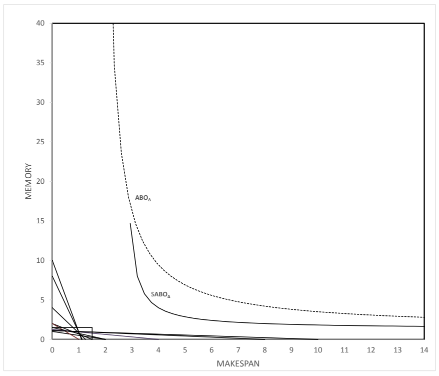
Algorithm	Approx. on makespan	Approx. on memory
$SABO_\triangle$	$(1+\triangle)\alpha^2\rho_1$ (Th. 10)	$(1+\frac{1}{\triangle})\rho_2$ (Th. 11)
$ABO_\triangle$	$(2-\frac{1}{m}+\triangle\alpha^2\rho_1)$ (Th. 12)	$(1+\frac{m}{\triangle})\rho_2$ (Th. 13)

Table 2: Summary of the results of $SABO_\triangle$ and $ABO_\triangle$.

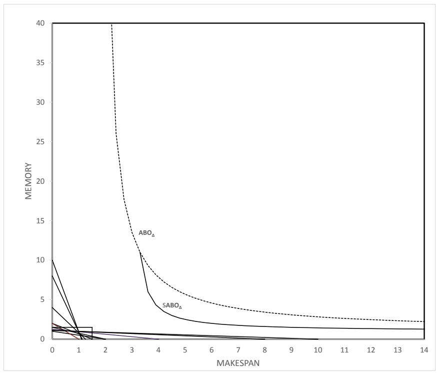
To better understand the tradeoff between memory consumption and makespan Figure 6 shows memory-makespan guarantees for the two algorithms. The bold lines shows impossibilities in the tradeoff between makespan and memory and means that no algorithm can guarantee better tradeoff than this. [22] discusses about these impossibilities in context of $SBO_\triangle$ algorithm.

The graph shows that for higher values for α the algorithm $ABO_\triangle$ have better tradeoff between memory-makespan than that of $SABO_\triangle$. For $\alpha\rho_1 \geq 2$, $ABO_\triangle$ always have better guarantee on makespan than $SABO_\triangle$. So, a schedule more centric to optimize makespan should follow $ABO_\triangle$ algorithm. And a memory centric schedule should follow $SABO_\triangle$ as the algorithm always has better guarantee on memory.

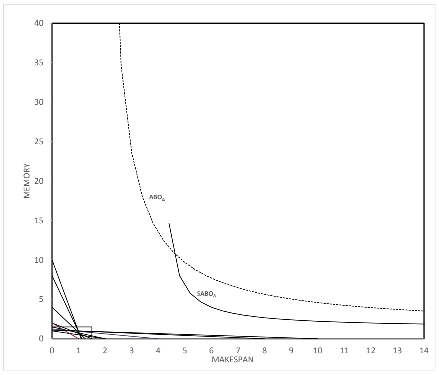
As a system designer, one always want to pick the algorithm and parameters that is best tradeoff between makespan and memory consumption. Depending on the guarantee, one should either pick $ABO_\triangle$ or $SABO_\triangle$ for scheduling tasks. For instance, if you want to guarantee a makespan less than 3 as in the case depicted in Figure 6b, you should use $ABO_\triangle$. However if you want a better guarantee on memory, you should always use $SABO_\triangle$ for task scheduling.



(a) $m = 5$, $\alpha^2 = 2$, $\rho_1 = \rho_2 = 4/3$



(b) $m = 5$, $\alpha = 3$, $\rho_1 = \rho_2 = 1$



(c) $m = 5$, $\alpha = 3$, $\rho_1 = \rho_2 = 4/3$

Figure 6: Memory-Makespan graph for $SABO_{\triangle}$ and $ABO_{\triangle}$. The bold lines represent impossibilities in tradeoff between guarantees.

CHAPTER 6: CONCLUSION AND FUTURE WORK

This thesis studies the effect on uncertainty in the processing time of tasks on scheduling for parallel and distributed machines. In particular, it investigates how allowing tasks to execute on different machines can help dealing with not knowing the processing time of tasks accurately. The thesis proposes three replication strategies, provides approximation algorithm in each case and a lower bound on the best achievable approximation in one of the case. Further to limit memory consumption the thesis presents two memory-aware bi-objective algorithms, one of which chooses only critical tasks to replicate and limits memory consumption.

The various strategies allow to trade the number of replication for a better guarantee. The results of these strategies show that a better guarantee can be achieved with fewer replication than that can be achieved by putting the data of a task on only one machine and even a small amount of replications can improve the guarantee significantly. These observations concludes that deploying the data on multiple machines can be an effective way of dealing with processing time uncertainties.

The bi-objective algorithms proposed in this thesis, schedule the memory intensive tasks and the processing time intensive tasks differently and optimizes both the objectives. One of the algorithms, chooses processing time intensive tasks to replicate and achieves better guarantee for higher values of α.

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.

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] P.-F. Dutot, K. Rzadca, E. Saule, and D. Trystram. *Multi-objective scheduling*,

chapter 9. Introduction to scheduling. Chapman and Hall/CRC Press, Nov. 2009. ISBN: 978-1420072730.

[8] 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.

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

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

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

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

[13] Z. Guo, G. Fox, and M. Zhou. Investigation of data locality in mapreduce. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 419–426, Washington, DC, USA, 2012. IEEE Computer Society.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

[21] B. T. Rao and L. S. S. Reddy. Survey on improved scheduling in hadoop mapreduce in cloud environments. *CoRR*, abs/1207.0780, 2012.

[22] E. Saule, P.-F. Dutot, and G. Mounie. Scheduling with storage constraints. *Parallel and Distributed Processing Symposium, International*, 0:1–8, 2008.

[23] V. T'kindt and J. Billaut. *Multicriteria Scheduling*. Springer, 2007.

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

[25] 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.

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

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

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

[29] 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.

[30] 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.

[31] 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.

[32] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, Apr 2009.

[33] 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 thef 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.

[34] 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.