PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/72556

Please be advised that this information was generated on 2018-01-11 and may be subject to change.
Techniques and Evaluation of Processor Co-allocation in Multi-cluster Systems

Een wetenschappelijke proeve op het gebied van de Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. mr. S.C.J.J Kortmann,
volgens besluit van het College van Decanen
in het openbaar te verdedigen op maandag 15 September 2008
om 13:30 uur precies

door

John Ngubiri
geboren op 1 oktober 1975
te Masaka, Oeganda
Promotores:
Prof. dr. Mario van Vliet
Prof. dr. ir. T. P. van der Weide

Manuscriptcommissie:
Prof. dr. H. A. Proper
Prof. dr. D. G. Feitelson (Hebrew University Jerusalem)
Prof. dr. V. B. Baryamureeba (Makerere University)
Dr. ir. D. H. J. Epema (Technische Universiteit Delft)
Preface

I always had the desire to undertake graduate studies. However, I had no funds. I was therefore convinced it will not be possible. In my final year of B. Sc, prof. L.S. Luboobi (current VC, Makerere University) told us that even without a lot of money, one can undertake graduate studies so long as he is serous. I did not take him seriously. After my B.Sc, I looked for a job and started working. After about one year, I realized I still had the desire for graduate studies. Classmates like Dr. Ssenyonga, Dr. Okullo, Justine Kasigwa and Martin Ngobye had enrolled already. I made up my mind to enroll the next year. Around that time, many organizations in Uganda were automating their systems. I felt I should be part of the 'revolution'. I therefore aimed at becoming a programmer. However, abandoning my passion of teaching, getting detachment from my idols like Mr. Kasozi and Mr. Kyalimpa and leaving friends like Ssebwunde and Musoke (in Masaka) made me a bit uncomfortable.

In 2001, I enrolled a Postgraduate Diploma in Computer Science. I met Dr. V. Baryamureeba (VB) who later became my supervisor. VB could offer maximum support so long as one was serious. I enjoyed working sometimes until the early morning hours under VB. Around the same time, Makerere University (Mak) had a project to make all staff computer literate. I got a part time job to teach computer literacy. This not only took me back to my passion of teaching but also linked me with the 'young' staff of Mak. These included Josephine, Habib, Julianne, Florence, Bob and Richard. Together with classmates like Sara, Winnie, Rogers, Fred, Maureen and Peace, life was interesting. The next logical step was to enroll for M.Sc in 2002. The M.Sc brought in more inspiration from senior teachers like Dr. Ssewanyana, Dr Tusubira and Dr. Muliira. Visiting staff like Dr Williams and Dr Mugisa, together with staff from Mathematics like Prof. Mugambi, Dr. Mugisha, Dr. Ssematya, Dr. Kasozi and Dr. Mango used to give guest lectures.
This enriched the program greatly.

Towards the end of my M.Sc. in 2004, I got appointed as a Teaching Assistant in Mak. That made my doctoral dream more realizable since the staff development policy at Mak is so 'staff friendly'. Unknown to me, the 'Nuffic NPT Project on Building a Sustainable ICT Training Capacity in the Public Universities in Uganda' was in its final stages of approval. It was to fund my Ph.D. in at Radboud University for four years.

I reached Nijmegen in fall 2004. The weather was cold (by Ugandan standards) and the sun was so rare. This highly affected my mood. However, my officemate (Drs. Ger Paulussen) seemed to understand my problem more than me. He was able to devise coping strategies for me. Other than weather, life in The Netherlands was very good. The academic set up was so good, professors like Theo van der Weide and Erik Proper provided the badly needed initial guidance and people like Nicole, Marijke and Wendy ensured 'you are not lost'.

Researching with/under Mario was an exciting experience. Mario has a sharp eye that can foresee likely huddles ahead. He therefore asks questions that bring up the whole gist of the matter. A meeting with Mario, in many cases, could leave me with a vision wider than what I hoped to explain to him prior to the meeting. Mario also linked me with other professors in and outside the Netherlands. It is impossible to list them all. Professors like D.H.J. Epema (T.U. Delft) and D.G. Feitelson (Hebrew University Jerusalem) were helpful beyond words can express. Correspondences with them gave me ideas, insights and perspectives that made my research interesting and rewarding.

My research would be impossible without the support of my family. I therefore take this opportunity to thank my wife Eva for all the support and understanding. She had to account for my absence to my (sometimes so inquisitive) son Matthew. Somehow she managed to keep reminding my daughter Vivienne that I exist. Every time I went back to home, I wasn’t a stranger to her.

Like earlier indicated, this research was fully funded by NUFFIC. I therefore thank the Dutch taxpayers for availing part of their tax to a Ugandan who wanted to enroll for graduate studies but lacked funds.

John Ngubiri, June 2008.
Contents

1 Introduction .......................................................... 1
  1.1 Background ...................................................... 1
    1.1.1 The demand for high computing power ..................... 1
    1.1.2 Multi-cluster systems and resource co-allocation .......... 3
  1.2 Parallel Job Scheduling ......................................... 5
    1.2.1 Parallel job scheduling algorithms ......................... 5
    1.2.2 The effect of information availability ...................... 8
    1.2.3 Research in multi-cluster co-allocation .................. 10
  1.3 Performance Evaluation ......................................... 15
    1.3.1 Performance metrics ....................................... 16
    1.3.2 Challenges with performance interpretation .............. 17
  1.4 Contribution ..................................................... 19
  1.5 Outline of the Sequel .......................................... 21

2 Problem Formulation .................................................. 23
  2.1 Introduction ..................................................... 23
  2.2 The Job Stream .................................................. 24
    2.2.1 Approaches to job stream generation ....................... 25
    2.2.2 Communication within the jobs ............................ 26
  2.3 The Multi-cluster System ........................................ 26
    2.3.1 The architecture ............................................ 26
    2.3.2 Queue and scheduler ....................................... 27
CONTENTS

2.4 Metrics, Constraints and Objectives . . . . . . . . . . . . . . . 27
  2.4.1 Performance metrics . . . . . . . . . . . . . . . . . . . 27
  2.4.2 The constraints . . . . . . . . . . . . . . . . . . . . . 27
  2.4.3 The objectives . . . . . . . . . . . . . . . . . . . . . 28
2.5 Simulation Environment . . . . . . . . . . . . . . . . . . . . . 28
2.6 Results Presentation . . . . . . . . . . . . . . . . . . . . . . 28

3 Group-wise Performance Evaluation 31
  3.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
    3.1.1 Workload mix and perceived performance . . . . . . . 31
    3.1.2 Groups and performance interpretation . . . . . . . 32
  3.2 Partition Heuristics & Group Generation . . . . . . . . . . . . 33
    3.2.1 Partition heuristics . . . . . . . . . . . . . . . . . . . 34
    3.2.2 Groups generation . . . . . . . . . . . . . . . . . . . . 34
  3.3 System and Job Stream Instances . . . . . . . . . . . . . . . . 35
  3.4 Scheduling Algorithm and Placement Policy . . . . . . . . . . 37
    3.4.1 Scheduling algorithm . . . . . . . . . . . . . . . . . . . 37
    3.4.2 Placement policy . . . . . . . . . . . . . . . . . . . . 37
  3.5 Group-wise Relative Performance . . . . . . . . . . . . . . . . 37
    3.5.1 Relative performance at fixed load . . . . . . . . . . . 38
    3.5.2 Performance variation with utilization . . . . . . . 39
    3.5.3 Implication of the results . . . . . . . . . . . . . . . . 39
  3.6 Performance Sensitivity to Parameters . . . . . . . . . . . . . . 43
    3.6.1 Sensitivity to $maxJumps$ . . . . . . . . . . . . . . . 43
    3.6.2 The effect of $thres$ . . . . . . . . . . . . . . . . . . 43
    3.6.3 Implications of the results . . . . . . . . . . . . . . . 45
  3.7 The Effect of the Partitioning Heuristics . . . . . . . . . . . . 46
    3.7.1 Random vs phased approach . . . . . . . . . . . . . . 46
    3.7.2 Implications of the results . . . . . . . . . . . . . . . 47
  3.8 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

4 The Greedy Multi-cluster Scheduler 49
  4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
CONTENTS

7.2.2 Metrics in the approach ............................................. 106
7.2.3 Example .......................................................... 108
7.3 Experimental Evaluation .................................................. 109
  7.3.1 Experimental set up ................................................. 110
  7.3.2 Results ........................................................... 111
7.4 Validation of the Approach ............................................... 113
  7.4.1 Performance and discrimination trends .............................. 113
  7.4.2 Deductions ....................................................... 114
  7.4.3 Circumstantial appropriateness of the metrics ...................... 116
7.5 Conclusion ............................................................ 117

8 Concluding Remarks ................................................................ 119
  8.1 Overview .............................................................. 119
  8.2 Contribution ........................................................... 121
    8.2.1 Performance Evaluation ........................................... 121
    8.2.2 Fairness Evaluation ............................................... 123
    8.2.3 Co-allocation Techniques ......................................... 124
  8.3 Future Research ........................................................ 125
    8.3.1 Schedulability estimation ......................................... 125
    8.3.2 The effect of communication ...................................... 126
    8.3.3 Communication cognizant scheduling ............................ 126
    8.3.4 Fairness evaluation ................................................ 127
Introduction

Outline: In our study, we address two aspects of processor co-allocation in multi-cluster systems: co-allocation techniques and scheduler evaluation. In this chapter, we discuss the need and practice of processor co-allocation. In Section 1.1, we give the background and need for processor co-allocation. In Section 1.2, we discuss recent related research in the field of parallel job scheduling in general and processor co-allocation in particular. In Section 1.3, we discuss the practice and challenges of performance evaluation for parallel job schedulers. We summarize the contribution of our study in Section 1.4 and outline the rest of the thesis in Section 1.5.

1.1 Background

1.1.1 The demand for high computing power

There are many computer applications that require a lot of computer resources. These include seismology, weather prediction and carbon chain analysis. There are also big advances in scientific research that require enormous computer resources. This led to more demand for high computational power [80]. This demand stimulated research in ways of increasing computational power. Over the past decades, therefore, computational power has experienced a rapid growth [79][80]. Researchers, in response to the growing computational power, took the advantage to venture into more resource
intensive research and applications. This resulted into a cycle; power intensive applications stimulate the growth in computational power and available computational power stimulates the development of more power intensive applications. Computer resources therefore, despite having a high growth rate, are still scarce. There is, as well, a natural quest for affordable computational resources in organizations. These factors call for techniques for optimal utilization of the available (computer) resources. Optimal utilization gives maximal satisfaction to the users/organizations in a cost effective manner. One approach to optimal utilization of resources is by making good schedules on resource usage within the organization. Another approach is for several organizations to collect all their computer resources into a common pool and share them. In this a set up, each organization has access to more computer resources than it procured. It can therefore process a job that cannot fit on its system alone. Alternatively, both approaches can be used. The organizations can agree to share the resources but also design good schedules such that all get a good share of resources to meet their computational needs.

Currently, high performance computing platforms include distributed memory multiprocessors, shared memory computers, massively parallel processors and clusters. Clusters form the most popular platform with over 80% of the top 500 supercomputers in the world [87]. A cluster is a group of loosely coupled computers that work closely together as a single computational facility. The computers in a cluster are connected by a fast local area network.

High performance applications are mostly presented as parallel jobs. A parallel job consists of a set of tasks/processes running concurrently to achieve a certain goal/objective. Each task runs on its processor. The number of tasks (and hence processors required) a certain job has is referred to as the job size. Since the tasks aim at a common result, they communicate as they execute. This implies that tasks of the same job have to run in parallel. In some applications, the size of the job can be adjusted while it is not possible in others. Jobs whose size can be adjusted are referred to as moldable jobs while those whose size cannot be adjusted are called rigid jobs. For moldable jobs, the adjustment in size leads to an adjustment in runtime [22].
1.1. BACKGROUND

Since tasks of a parallel job have to run simultaneously, execution of a parallel job requires a simultaneous availability of computer resources for all the tasks. Common resources include processors, memory, bandwidth, specialized software, specialized data and time. A job only executes if the required resources are available in sufficient quantities. The resources, however, may not always be available in sufficient quantities to satisfy all available jobs. Jobs, therefore, have to be queued and the scheduler picks the jobs to be executed as resources get available. There is a need for good scheduling policies in order to give every job/user the best satisfaction possible. Ideally, all resources should be available in sufficient quantities for a job to execute. However, some jobs may be able to execute with slight shortfalls in some of the resources. Resources can therefore be looked at as not being equally important. When processing rigid jobs, for example, a shortfall in memory may reduce the rate of processing. However, a shortfall in processors available completely impedes the processing. In hard real-time systems, time is a very important resource. This is because the value attached to the job output falls drastically after its deadline is passed. In systems like gang scheduling [68], memory is more scarce compared to cases of pure space slicing. This is because in gang scheduling, several jobs can execute on the same set of processors by having tasks of each job execute for a pre-determined time interval. Processing multiple jobs on the same processor can deplete its memory. Some data is therefore written to disk leading to lower processing rates. This translates to poorer performance [6].

1.1.2 Multi-cluster systems and resource co-allocation

Clusters are currently a dominant supercomputing platform. The dominance can be attributed to their cost effectiveness and scalability. Recently, some research work (like in [5]) has been done on merging clusters into larger computational facilities called multi-cluster systems (mini-grids in some literature). This fits in earlier research works in computational and data grids [36][83]. Examples of existing multi-cluster systems include the Distributed
CHAPTER 1. INTRODUCTION

ASCI Supercomputer (DAS) [82] owned by the Advanced School of Computing and Imaging (ASCI) in The Netherlands and the Clemson Computational Mini-grid Facility [81] located at Clemson University, USA.

Though multi-cluster systems may be looked at as a small version of the conventional grid, they have operational differences other than the smaller size. Clusters in multi-cluster systems are connected by a dedicated backbone. In case of the conventional grid, the connection is by Internet. This implies that multi-cluster systems have a more reliable bandwidth that connects the participating clusters/resources. The reliable bandwidth ensures reliable accessibility to remote clusters by users. This makes resource co-allocation [21] more feasible in multi-cluster systems than on the conventional grid.

In resource co-allocation, a parallel job is broken into components and each component can be processed in a different cluster. Logically, co-allocation brings together resources which are distributed in different clusters so as to process a resource intensive job. In doing so, resource fragmentation is reduced and system utilization is increased. Since the components belong to the same job, the processes communicate during execution. The progress of one process, for example, may need data from another process. This implies that inter-cluster and intra-cluster communication of co-allocated jobs is inevitable. This further dictates that the different components of co-allocated parallel jobs have to start (and end) processing at the same time. Despite the benefits, co-allocation comes with practical setbacks. The setbacks are:

(i) Jobs’ suitability for co-allocation:

The parallel jobs may be in such a way that they are not suitable for co-allocation. There may be so much communication among remote tasks which may (over) saturate the inter-cluster links. This may impede continuation of processing of the co-allocated job. Even with a non saturated link, the slower inter-cluster links may excessively increase job execution time leading to poor performance.

(ii) Scheduler appropriateness:

The schedulers need more information in cases co-allocation than they
1.2. PARALLEL JOB SCHEDULING

would need in case of a single cluster. The extra information needed include resource availability in remote clusters, inter-cluster link states, cluster owners policies and others. Single cluster schedulers therefore need to be enhanced to handle co-allocation.

Co-allocation comes with benefits of bringing together scattered computer resources. However, the scheduler needs enhancement to cope with the demands that come with co-allocation. Moreover, the communication characteristics of the jobs may cause excessive deterioration in performance when co-allocated. Co-allocation, therefore, is not always viable.

1.2 Parallel Job Scheduling

Parallel job scheduling has been an active field of research for over a decade [26]. Most of the research has been done in scheduling techniques, scheduler evaluation, workload modeling and fairness. The platforms considered include shared memory computers, distributed memory multiprocessors, clusters, multi-cluster systems and the grid. Broadly, parallel jobs are scheduled to optimize a certain objective subject to some constraints. The objective to optimize is the performance metric used in the system. This can be job average waiting time, throughput, average job slow down, etc. The constraints can be memory, processors, time or (specialized) software/data.

Studies in fairness show that an optimal metric does not necessarily imply a better scheduler. This is because it can be a result of favoring certain jobs at the expense of others. The metric value obtained therefore needs further interpretation so as to come up with deductions which represent user satisfaction in practice. Parallel job scheduling therefore goes beyond optimizing the performance metric to what would be realistically acceptable.

1.2.1 Parallel job scheduling algorithms

Several parallel job scheduling algorithms have been proposed. The heuristics employed by the algorithm highly depend on the set up of the system. Some
of the factors in the set up that affect schedulers include:

(i) **Possibility of time slicing:**
If time slicing is possible, then the number of jobs a certain processor can accommodate increases. Techniques like gang scheduling [68] can also be employed. However, the memory on a processor becomes a delicate resource. Having many jobs simultaneously running on a processor can deplete its memory. This forces the computer to first write the data to disk. Processing a job becomes more expensive since it involves disk access.

(ii) **Possibility of migration:**
If tasks of a running job can be transferred from one cluster to another, then the system can be able to reallocate running jobs to create space for waiting jobs. Migrating jobs [24] can improve the packing of jobs and hence performance. In some cases, a combination of different approaches [97][98] can be used to improve system performance.

(iii) **Availability of Information:**
In some cases, some of the information on the job may not be known in advance [23]. This limits the extent to which schedulers make decisions. Unknown information may be job execution time, memory requirements and communication intensity. The unknown information may greatly influence the performance of the system if in case the system underestimates/ignores them.

In this work, we limit ourselves to dedicated pure space slicing systems with no migration.

**Backfilling**
For space slicing schedulers, the most prominent scheduling approach over the past decade is backfilling [26]. Backfilling improves performance in the system by allowing small jobs leapfrog and get processed outside their queuing order. The leapfrogging jobs utilize processors that would be idle if jobs
were to be processed in the strict queuing order. It reduces fragmentation and capacity loss leading to higher utilization and better performance.

Backfilling has some implementation variants. Lifka [57] proposed the *conservative* approach to backfilling. In this approach, a job is allowed to jump only if it will not delay the reservation time of any of the jobs ahead of it in the queue. Mu’alem and Feitelson [58] proposed the *aggressive* approach to backfilling. In this approach, only the reservation for the job at the head of the queue is made. A job can be made to leapfrog and get processed so long as it does not delay the starting time of the job at the head of the queue.

Conservative backfilling reduces the number of possible jobs that can be able to utilize an existing processor hole. This implies that more holes are left unutilized in conservative backfilling than in aggressive backfilling. Aggressive backfilling offers more opportunities for utilizing the free processors but it can lead to delay of jobs close to but not yet at the head of the queue. Both approaches lead to performance improvements compared to FCFS.

The performance differences between aggressive and conservative backfilling largely depend on the workload used. Srinivasan et al. [77][78] showed that there are big performance differences among size-based different job groups. The dominating job group dictates the overall relative performance of the two backfilling approaches. To improve the entire job stream performance, it is necessary to strike a balance between conservative and aggressive backfilling. This leads to flexibility in aggressive backfilling and reduces the resources spent in computing reservations for the entire job stream in conservative backfilling. Studies by Chiang et al. [18] showed that while it is common for a backfilling job (in aggressive backfilling) to delay jobs other than the first in the queue, it is rare to delay jobs so deep in the queue. Making reservations for the first 3 to 6 jobs makes a better compromise.

**Enhancements on backfilling**

Some research was carried out to enhance backfilling to improve performance further. This was mostly focused on the choice of the job to backfill and choice of the job(s) whose reservation(s) is/are made.
Both aggressive and conservative backfilling pick the first job that satisfies the backfill condition. Such a job does not necessarily make optimal utility of the available processor hole. Shmueli and Feitelson [73][74] proposed the backfilling with lookahead approach. In this approach, the scheduler looks ahead into the queue for the job that best utilizes the existing processor hole. To reduce complexity, the best fitting job has to be searched for up to a certain depth into the queue. The lookahead technique can be applied to conservative or aggressive backfilling. It puts residual processors to optimum utility. This minimizes capacity loss and leads to better performance.

Srinivasan et al. [78] proposed an approach where reservations for jobs are made selectively when backfilling. A job only gets a reservation if there is evidence that it is tending to excessive starvation. For every job, the eXpansion Factor (XFactor) is computed ($XFactor = \frac{(\text{Wait time} + \text{Estimated Run Time})}{\text{Estimated Run Time}}$). A job is given a reservation only if its $XFactor$ is greater than a certain threshold. This protects jobs from excessive starvation when the system still enjoys the flexibility of aggressive backfilling.

### 1.2.2 The effect of information availability

Backfilling assumes knowledge of job characteristics like size and duration. In some cases some of the information is unknown [23]. This limits the extent to which the scheduler can make decisions with certainty. In some cases, parameters are inaccurately estimated by the users. Studies by Lee et al. [54] showed that users are unable to accurately estimate their job runtimes. This is valid even in cases where there is no termination of jobs executing beyond a certain time threshold. Inaccurate runtimes do not generate accurate reservation times in backfilling. Chiang et al. [18] studied the effect of inaccuracies in runtimes on the performance of backfilling. Jones [47] studied the effect of inaccuracies in estimates of job bandwidth requirements on the performance of co-allocation. Generally, inaccurate estimates of job parameters lead to poorer scheduler performance. The scheduler, therefore, needs to have mechanisms to handle the problem of inaccurate parameter
estimates. Jones [49] used checkpointing to recover from wrong decisions by a multi-cluster scheduler in a system that allows co-allocation. Wrong decisions were due to unknown bandwidth requirements. If the scheduler realized that a running job has a very big bandwidth deficit (implying it was wrong to co-allocate it), the job would be terminated and rescheduled without co-allocation. This approach leads to performance improvements. However, since the job is first processed, terminated and then reprocessed, some processor hours were wasted.

Another approach is for the scheduler to estimate the unknown parameters.Schedulers can use historical data to make estimates of the jobs which are yet to be processed [93]. These approximations, though with some inaccuracies, makes backfilling possible. Tsafrir et al. [95] showed that system generated runtime estimates give better performance compared to user runtime estimates. Some studies [33][58][99] showed that inaccuracies in user runtime estimates can actually be of performance benefit. This is not to imply that users are free to wrongly estimate their runtimes and get good performance from the system [92][94]. Tsafrir et al. [94] observed that actually the performance improvement was due to the relationship between the inaccurate runtime estimate and the job kill time. It was not due to the relationship between the actual and estimated runtime (error). Having a more accurate runtime estimate therefore is of paramount importance for good performance in backfilling.

Runtime estimates are mostly point estimates. Definitely, it is hard to have an accurate point estimate. Nissimov and Feitelson [67] proposed using a range rather than a point estimate. Range rather than point estimates make the actual runtime easier to capture and hence make more accurate reservations. However, the reservations are also probabilistic which calls for improvements on the ways reservations are handled.

Another approach of dealing with inaccurate/unknown parameters is by not putting the unknown parameters (like job runtimes) into consideration when making scheduling decisions. The scheduler concentrates on what it knows to make scheduling decisions. The scheduler, in such a case, has to
devise other ways of checking the aspects the unknown parameter(s) would have checked. Jones [49], for example, used check pointing to cater for unknown bandwidth requirements by the job. Aida et al. [2] used the number of time a job at the head of the queue is jumped in Fit Processors First Served algorithm (FPFS) to control starvation other than job reservations that would be used in backfilling if the runtimes were known.

In this work, we consider a case where the job runtime is unknown and the system makes no attempt to estimate it. However, all job runtimes are known to be finite. The system therefore does not kill any job due to excessive execution.

1.2.3 Research in multi-cluster co-allocation

Research in multi-cluster co-allocation is more recent when compared to that in general parallel job scheduling, computational or data grids [26][83]. Substantial research work has been done on DAS [5][82] and the Clemson Computational Mini-grid Supercomputing Facility [48][81]. Co-allocation has to address the challenges of multi-site scheduling as well as optimal usage of the participating clusters. This calls for improvement of single cluster algorithms to cater for the new challenges.

Bucur [12] evaluated the performance of multi-cluster processor co-allocation. She mostly considered differences in architectures, job structures, communication, component distribution, partition approaches and job request policies. Jones [46] studied ways the dynamic nature of network resources affect the performance of co-allocation, looked deeper into network resources management, and investigated the effect salient job stream characteristics (like bandwidth) on performance and fairness.

Multi-cluster architectures

Multi-cluster architectures vary in (i) the number and relative size of clusters, (ii) number, scope and priority of queues and (iii) number and roles of schedulers. The architectural differences lead to differences in the way the
1.2. PARALLEL JOB SCHEDULING

entire job stream and specific jobs perform.

Bucur and Epema [8] studied the effect of system configurations on performance of co-allocation. They showed that:

(i) It gives performance benefit to have a multi-cluster system of equal clusters instead of having clusters of different sizes;

(ii) Having a system of fewer big clusters leads to better performance compared to a case of many small clusters; and

(iii) Different component placement policies perform differently depending on the number and size of clusters. In case of a system with few large clusters, Worst Fit (WFit) is a better placement policy. In case of a system has many small clusters, First Fit (FFit) is a better policy.

Their findings show that on top of the scheduling techniques, architectural and placement considerations can improve the performance of co-allocation.

A multi-cluster system may also have different queue/scheduler configurations. Possible queue/scheduler configurations include (i) all clusters are served by a single queue and scheduler, (ii) each cluster has a queue and scheduler and (iii) each cluster has a queue and scheduler for (local) single component jobs while the entire system has a global queue and scheduler for multi-component jobs.

Bucur and Epema [13][14] showed that for multi-component jobs, having many schedulers and distributing the multi-component jobs evenly among them gives a performance benefit. They also showed that in cases where there are separate queues for single and multi-component jobs, favoring multi-component jobs leads to poorer performance. In case multi-component jobs are given a higher priority, it is of performance benefit to allow single component jobs to jump non-fitting multi-component jobs and utilize available processors. This is valid even when such jumps may delay the multi-component job a bit more. They further showed that equally distributing multi-component jobs among the queues brings better performance than having them put in a specialized queue. In cases where jobs local to the cluster are put in a single queue, it is better to give them higher priority since
the components of multi-component jobs have chances of being processed in other clusters.

Communication

Communication is an important aspect of co-allocation. Possibly, it constitutes the biggest operational difference between a multi-cluster system and a single cluster of the same number of processors. It is mostly communication that makes co-allocation viable or unviable. Inter-component communication across inter-cluster links increases job execution times. Jobs therefore occupy the processors for a longer time. The extra occupation of the clusters by a co-allocated job does not actually lead to processing extra data. It brings in the concept of net and gross time spent on the processors. It also brings in the concept of net and gross utilization of the system [15]. Though the extra time is sometimes avoidable, it is not necessarily beneficial to avoid it. The processor time saved can as well be lost to system fragmentation.

Communication in multi-cluster systems was studied in works like [40][41][42][43][44]. Its effect on the performance of co-allocation was also studied in [7][9][50]. Communication in a parallel job may be synchronous or asynchronous. In synchronous communication, the execution of a parallel job is made up of successive communication and processing phases. Each phase can only start if the phase preceding it has completed. In asynchronous communication, job processes run independently but any two may randomly communicate.

The nature of communication in a job determines how it is affected by co-allocation. In synchronous communication, so long as the job is broken up into at least two components, the execution time is independent of the number of components. This is because inter and intra cluster communication takes place simultaneously. For the asynchronous case, increasing the components increases the number inter-cluster communication messages. Since the communication does not take place in parallel, it leads to more time spent in communication and hence job execution time.

Bucur and Epema [7][9] considered a synchronous communication model
where all co-allocated jobs suffer a fixed execution time penalty. They used FCFS scheduler and investigated the intra-cluster to inter-cluster speed ratio beyond which co-allocation is not viable. Sonmez et al. [75] studied an asynchronous communication model and proposed component placement heuristics that reduce the effect of communication. The heuristics reduce the execution time penalty suffered due to communication. The heuristics aim at minimizing the number of clusters a job is processed in. This is done by breaking up the jobs into components that fully fit in the largest available processor holes. In case the number and size of the components are predetermined, possibilities of processing multiple components in one cluster are explored. If they exist, they are exploited.

Jones et al. [50] studied the effect of communication using the concept of bandwidth saturation. If a job is allocated the bandwidth required, it processes and finishes within its allotted execution time. However, if there is a shortfall in bandwidth, the rate of processing reduces by a value proportional to the shortfall. This leads to an increase in the job execution time. The bandwidth shortfall may not last for the entire execution time of the job. In some cases, the job may suffer different shortfalls in different time intervals. The execution time penalty is therefore computed step-wise depending on link states. Further studies by Jones [47] were carried out on the effect of inaccurate bandwidth estimates.

Though co-allocation has challenges especially with communication, it is viable within some parameter ranges. Likewise, with good techniques (like in [45][75]) the effect of communication can be minimized and more performance improvements obtained from the packing benefits of breaking up large jobs.

**Job streams**

Performance of co-allocation (and parallel job scheduling in general) highly depend on the workloads used [8][11][31]. In some cases, the results are more dependent on the job streams than the scheduler [27]. Studies by Frachtenberg and Feitelson [37] showed that using unrealistic job streams is a major pitfall committed when evaluating parallel job schedulers. It is
therefore of utmost importance that realistic job streams are used.

Job stream characteristics include job size, communication pattern/intensity, memory, arrival patterns, self-similarity, runtime and I/O. Co-allocation is mostly studied by simulation. There is therefore a big need to realistically represent these characteristics in the simulated system. Job streams can be generated synthetically or read from logs of existing supercomputing sites.

Synthetic workloads are obtained by statistically generating the job stream characteristics. These distributions are obtained from studies on traces from existing supercomputers [19][56]. The advantages of synthetic workloads are that they are easy to generate, adjust and extrapolate. The disadvantage however is that it is hard to get a correct combination of the distributions that accurately represent the workload [31]. For example, the size distribution may be correct while runtime and arrival pattern distributions are not. This may lead to a wrong job stream and hence wrong deductions [27].

Another option is to use archived logs [82][83] from existing supercomputers. These logs are advantageous since a real job stream can be obtained without having to know the distributions of parameters that generate it. However, using traces also have some setbacks which include:

(i) **Hardness to extrapolate:**
   In some cases, the archived trace may be too short to generate a steady state in the simulated experiments. Since the distributions of the job parameters are unknown, the job stream is hard to extrapolate.

(ii) **Missing information:**
    In some cases, not all details are archived. This implies that the trace can only be used in cases where the missed information is not necessary. Parameters like communication are rarely archived. In a study of the DAS trace [56] for example, each cluster was studied separately hence data on co-allocation was not included.

(iii) **Hardness in adjustment:**
    In many cases, the simulated system and the actual system from which the logs were got do not have the same size. This may lead to jobs that
cannot fit in the simulated system. Using only jobs below a certain size does not necessarily generate the would be job stream if the system was smaller. In other cases, there may be a need to adjust the load in the system. Changing parameters like inter-arrival time alters the daily/weekly traffic peaks. This would actually not be the case if the load was different in real life.

(iv) Workload flurries:
There is a tendency of users submitting the same job in a small interval of time (workload flurries). These flurries greatly affect the results from the job stream [91]. This calls for workload sanitization [28].

Both synthetic and real workloads have merits and demerits. Most researchers use both. When comparing schedulers, workload traces are mostly used and when studying sensitivity to parameters, synthetic workloads are used. In this thesis, we use both synthetic workloads and traces.

Co-allocation techniques

Co-allocation calls for improvement of the general parallel job schedulers in order to incorporate multi-component jobs. Bucur and Epema [10] proposed and evaluated several policies for different architectures. On top of the scheduling policies, schedulers also use heuristics for mapping components onto clusters (placement policies). They showed that different policies perform differently in different settings. More research was done on communication modeling [40][42][43][50], investigation of the viability of co-allocation [7][47] as well as minimizing the negative effect of co-allocation [45][75].

1.3 Performance Evaluation

Performance metrics are mostly used to compare schedulers [26]. The expectations of the users are used to derive a performance metrics. The metric is then used to evaluate scheduler performance.
1.3.1 Performance metrics

In parallel job scheduling, performance metrics can be classified into system and user based metrics. From the system perspective, we look at how optimally the scheduler uses the system resources. A scheduler that wastes resources is considered to be poor. System based metrics include system utilization ($\rho$) and capacity loss ($\epsilon$). System utilization refers to the average proportion of the system resources that are used to process the jobs. A system loses capacity if (i) it has jobs waiting in the queue to execute and (ii) it had free processors but, because of fragmentation, it cannot execute the waiting jobs. From a user’s perspective, we use the user’s expectation from the system to generate the metric. User based metrics include Average Waiting Time ($AWT$), Average Response Time ($ART$), Average Job Slow Down ($AJSD$) and Bounded Average Job Slow Down (BAJSD).

Let us consider a system made up of $N_p$ processors processing $N$ online jobs $J_1, J_2, \ldots, J_N$. Job $J_i$ has size $n_i$, execution time $t_i^e$, arrives at $t_i^a$, starts execution at time $t_i^s$ and finishes execution at time $t_i^f$ (for non time slicing processing, $t_i^e = t_i^f - t_i^s$). We assume that the first job $J_1$ arrives at time 0 and the last job to finish execution is $J_l$. A scheduling event takes place whenever a new job arrives or an executing job terminates (note that none, one or more that one scheduling event can happen when an executing job terminates/arrives). Let us assume that there are $n$ scheduling events where the $k^{th}$ event takes place at time $\phi_k$. The number of free processors after the $k^{th}$ scheduling event is $e_k$. We define a parameter $\delta_k$ to be 0 if there are no jobs in the queue after the $k^{th}$ scheduling event and 1 if there is at least a job in the queue after the $k^{th}$ scheduling event. We define the metrics:

$$\rho = \frac{1}{N_p \times t_l^f} \sum_{i=1}^{N} (n_i \times t_i^e)$$  \hspace{1cm} (1.1)$$

$$\epsilon = \frac{1}{N_p \times t_l^f} \sum_{i=1}^{n-1} e_i(\phi_{i+1} - \phi_i)\delta_i$$  \hspace{1cm} (1.2)$$

$$AWT = \frac{1}{N} \sum_{i=1}^{N} (t_i^a - t_i^s)$$  \hspace{1cm} (1.3)$$
1.3. PERFORMANCE EVALUATION

\[ ART = \frac{1}{N} \sum_{i=1}^{N} (t_{fi} - t_{ai}) \] \hspace{1cm} (1.4)

\[ AJSD = \frac{1}{N} \sum_{i=1}^{N} \left[ \frac{t_{fi} - t_{ai}}{t_{ei}} \right] \] \hspace{1cm} (1.5)

\[ BAJSD = \frac{1}{N} \sum_{i=1}^{N} \left[ \frac{t_{fi} - t_{ai}}{\max(t_{thrs}, t_{ei})} \right] \] \hspace{1cm} (1.6)

The time \( t_{\text{thrs}} \) is a predefined time which is in such a way that a job \( J_k \) where \( t_{k}^e < t_{\text{thrs}} \) is considered very short. It is used to eliminate the exaggerated effect of very short jobs on \( AJSD \).

Though performance metrics appear straightforward, in some cases, they may misrepresent the actual perception of the scheduler performance. This may lead to system owners not getting a true picture of users’ satisfaction.

1.3.2 Challenges with performance interpretation

Choosing a metric to use when evaluating parallel job schedulers is a hard decision to make [37]. This is because sometimes performance may be misrepresented. The misrepresentation can be due to:

(i) Appropriateness of the metric:

Some metrics may be inappropriate in some setups. This is because the metric does not represent the feeling of satisfaction by the user (job owner). For example, if average waiting time is used to measure performance for a real-time scheduler, then the metric is looking at performance from a different perspective compared to the user. A job may wait for a short time, but gets processed beyond its deadline. The metric implies a good performance while the user feels it is not.

(ii) Aggregation problem:

Some times, the metric may be a good representation of the user’s needs but special cases distort the aggregated metric. For example, \( JSD \) is a good metric as it relates the job runtime with the waiting time. However, short jobs tend to exaggerate their slow down which
is not in line with what the user feels (a job of runtime 0.2 seconds and wait time of 10 seconds has a slow down of 51). Computing the job stream average makes the $AJSD$ value misleadingly high for a job stream with many short jobs.

\( (iii) \) Concealment of starvation:

Some times, a small section of jobs get starved by the scheduler. However, since they are a minority, the starvation is not adequately implied by the job stream mean metric value. Previous studies in supercomputer workloads \[19\][56][58] show that small jobs make up the numerical majority and load minority in the job stream. A scheduler that favors small jobs over large jobs appears to perform better since starving one large job creates space for several small jobs. This is despite the fact that the majority of the workload is performing poorly. There may be a need to study performance in groups if a true picture of scheduler performance is to be got.

\( (iv) \) Interference from foreign factors:

In some cases, the perceived improvement in performance is due to foreign factors which may not be of actual performance benefit. For example, a co-allocated job’s execution time is longer than it would be on a single cluster. Since co-allocated jobs spend longer on the clusters, they may give an impression of higher system utilization. This increase in utilization is not always of benefit since it does not necessarily imply a higher rate of workload processing. Never the less, it is not necessarily a waste since the extra utilization could as well be lost as capacity loss.

When evaluating schedulers, care has to be taken that the implied improvement in performance is actually to the benefit of the users/system. We need to intuitively interpret the metric value implications in practice. Short of this may lead to metric values that misrepresent performance.
1.4 Contribution

The main contributions of our work are:

(i) **Group-wise performance evaluation:**
We study group-wise performance of processor co-allocation in multi-cluster systems. Our work extend previous studies that used job groups (like [76][77][78]). While these studies were carried out on single cluster systems, we do our study on a multi-cluster system. We also use three parameters to group up the jobs instead of one used in previous research. We study performance sensitivity of the different groups on selected scheduler and job stream parameters. We deduce the interpretational implications of using job stream average (performance) metric values other than job groups on user satisfaction.

(ii) **Relating jobs’ characteristics with schedulability:**
We study how jobs’ physical characteristics (size, number of components and width of the widest component) affect their schedulability in a multi-cluster set up. We propose ways performance of co-allocation in multi-cluster systems can be improved by manipulating job schedulability within the job stream.

(iii) **Proposition of the greedy multi-cluster scheduler:**
Motivated by (i) big differences in performance between large and small jobs and (ii) differences in job schedulability, we propose the greedy multi-cluster scheduler. The greedy scheduler gives a priority advantage to unschedulable jobs to increase their chances of being scheduled. We carry out parametric sensitivity studies of our scheduler. We also study group-wise and job stream performance bounds of our scheduler with FPFS scheduler. We show that our scheduler outperforms and is fairer than FPFS.

(iv) **Communication effect on the performance of co-allocation:**
We use the combination of intra-cluster to inter-cluster speed ratio and job communication intensity to model the effect of communication on
the performance of co-allocation. This extends previous studies that considered only intra-cluster to inter-cluster speed ratio [7]. We investigate the viability of co-allocation when communication is considered. Our investigation of viability boundaries is done using job groups which extends the boundaries got using the entire job stream. Our two parameter representation of communication effect also extends the interpretation of the boundaries of co-allocation viability as a function of job stream and system characteristics. We study the effects of selected job stream parameters as well as communication heterogeneity within the job stream on performance and viability of co-allocation.

**(v)** *Fairness metrics and representation of starvation:* Fewer studies (like [39][71][96]) have been carried out on fairness in parallel job scheduling. We examine the suitability of the existing fairness metrics as a means of measuring discrimination/favoritism in parallel job schedulers. We investigate whether the implied unfairness by selected (commonly used) metrics is always due to discrimination of jobs. We show that it is not. We identify cases where the implied unfairness by the metrics is not unfairness in practice.

**(vi)** *Proposition of the net benefit approach to fairness evaluation:* Since we showed (in (v) above) that some of the unfairness implied by the metrics is not actually unfairness in practice, we propose a new approach to evaluate fairness. Our approach bases on the net benefit a job gets in being scheduled by one scheduler other than another. We derive some metrics for this approach and compare selected multi-cluster schedulers for fairness using our metrics. We use the worst performing and most discriminated jobs in the job stream to interpret and validate our approach. We show that our approach is realistic.

In a nut shell, we study ways processor co-allocation is evaluated. This is by way of performance and fairness. We also study how communication characteristics of jobs affect quality of service got from the schedulers. These include physical (size, number of components and width of the widest com-
ponent) and intrinsic (communication intensity). We also study multi-cluster co-allocation techniques so as to improve performance and fairness.

1.5 Outline of the Sequel

We now briefly explain the content of each of the remaining chapters:

- **Chapter 2**: In Chapter 2, we formulate the general problem being addressed throughout our work. This includes the set up of the multi-cluster system, the model(s) of the job streams used and the way results are presented. We also include the methodology used and the general assumptions made.

- **Chapter 3**: In Chapter 3, we study group-wise performance of processor co-allocation in multi-cluster systems. We group the jobs by size, number of components and width of the widest component. We study the relative performance of the different groups and deduce how the different characteristics affect job schedulability. We also study the sensitivity of the different groups to selected job stream and scheduler parameters. Finally, we study how the mechanisms of partitioning large jobs into components (prior to co-allocation) affect scheduler performance.

- **Chapter 4**: In Chapter 4, we propose the greedy multi-cluster scheduler. The greedy scheduler separates the queuing order from the priority order. On top of the time spent in the queue, a job is given a priority boost proportional to how hard it is to schedule. This gives more scheduling opportunities to unschedulable jobs. This improves the packing scheme, increase utilization and leads to better performance. We perform sensitivity studies of the scheduler on its parameters as well as selected job stream parameters. Finally, we study the performance bounds of the greedy scheduler with FPFS.
CHAPTER 1. INTRODUCTION

• Chapter 5: In Chapter 5, we study the effect of wide area (inter-cluster) communication on the performance/viability of co-allocation. We use the intra-cluster to inter-cluster communication speed ratio and job communication intensity to model the runtime penalty due to co-allocation. We investigate the bounds within which co-allocation is viable. We study how the viability bounds vary with selected job stream parameters. We also study the effect of communication heterogeneity within the job stream on the viability/performance of co-allocation.

• Chapter 6: In Chapter 6, we study the concept of fairness in parallel job scheduling. We study the ways fairness has been evaluated in earlier research (like [39][71][96]) and the validity of deductions derived from the fairness metrics. We identify cases where deductions from fairness metric misrepresent the discrimination or favoritism the user gets from the system. We identify/highlight possible cases where implied unfairness may not be unfairness in practice.

• Chapter 7: In Chapter 7, motivated by the weaknesses of fairness metrics identified in Chapter 6, we propose the net benefit approach to fairness evaluation for parallel job schedulers. Our approach uses the net benefit a job gets by being scheduled by one scheduler instead of another to evaluate scheduler fairness. We use performance and discrimination trends to validate our approach. We propose fairness metrics based on our approach and use it to compare selected multi-cluster schedulers for fairness using our approach. We deduce the most appropriate metric.

• Chapter 8: Finally in Chapter 8, we summarize the results obtained in our work. We discuss our findings and make suggestions for research that can come after this study.
Chapter 2

Problem Formulation

Outline: In this chapter, we describe the general formulation of the problem addressed in our study. In Section 2.1, we highlight some of the possible variants of the set up. In Section 2.2, we describe the job streams to be used in the study. We then describe the multi-cluster set up in Section 2.3. In Section 2.4, we describe the performance metrics used for our evaluation. We also describe the general objectives and constraints. We describe the simulation environment of our studies in Section 2.5. Finally, we describe the presentation of results in Section 2.6.

2.1 Introduction

The problem of processor co-allocation can be formulated in different ways. The ways it is formulated determine the techniques that can be applied. At the same time, it determines the way the system can be evaluated. The differences in the way formulation is done can be due to factors like:

(i) Workload characteristics:

Though a lot of work has been done in workload modeling, there is no universally agreed way of generating typical supercomputer workloads. Even in cases where the workloads used are from archived logs of existing computers, deductions got are, sometimes, dependant on the source. Other workload related challenges include varying of the load generation of components prior to co-allocation.
CHAPTER 2. PROBLEM FORMULATION

(ii) Communication pattern and modeling:
Communication is an important aspect of co-allocation. This is because it largely determines co-allocation viability. The pattern and intensity of communication within the job stream has a big influence on the way the jobs can be co-allocated.

(iii) Cluster characteristics:
Since the multi-cluster system is made up of different (independent) clusters, the (relative) characteristics of the clusters can vary. The characteristics can include the sizes, the inter-cluster link speeds, the intra-cluster networks and the mode of processing. Different systems may have different cluster characteristics.

(iv) Queues and schedulers:
Multi-cluster systems can have different queue and scheduler configurations. For example, all clusters can be served by one queue and one scheduler. Alternatively, each cluster can have a queue and a scheduler. In other cases, there can be a global queue and scheduler that handles multi-component jobs while the local schedulers handle single component jobs.

The decisions made, and sometimes the results obtained depend on the set up of the multi-cluster system considered.

2.2 The Job Stream

We consider online job streams. Jobs randomly arrive into the queue from which they are picked by the scheduler and get allocated to the clusters. The characteristics of a job that has not yet arrived are unknown to the scheduler. When a job arrives, its size gets known to the scheduler. The runtime is known to be finite but its value is unknown to the scheduler until it has finished execution. Job sizes cannot be varied. Any job can be broken into components and be co-allocated. If a job is co-allocated, its runtime
2.2. THE JOB STREAM

increases. The increase in runtime is caused by the relatively slower inter-cluster links.

2.2.1 Approaches to job stream generation

We use two approaches to job stream generation. One approach is by synthetically generating the job stream characteristics (like size, runtime and inter-arrival time) from statistical distributions. The second approach is by using job stream characteristics from archived supercomputer workloads.

The synthetic approach

In the synthetic approach, we generate job sizes from the distribution $D(q)$ defined over an interval $[\bar{n}, \bar{n}]$ ($0 < \bar{n} < \bar{n}$). In $D(q)$, the probability $p_i$ that a job has a size $i$ is given by

$$p_i = \begin{cases} \frac{3q^i}{Q} & \text{if } i \text{ is a power of 2} \\ \frac{q^i}{Q} & \text{if } i \text{ is not a power of 2} \end{cases}$$

(2.1)

The parameter $q$ ($0 < q < 1$) is used to vary the average job size while the parameter $Q$ is in such a way that $p_i$ sums up to 1. This distribution favors small jobs and those whose size is a power of 2 which is known to be a realistic choice [30]. Job runtimes and inter-arrival times are generated from an exponential distribution. The mean runtime is 10. These job size, runtime and inter-arrival distributions were also used in previous related research (like [10][12]).

Using archived traces

In this approach, we use traces from the parallel workloads archive [86] and the grid workloads archive [84]. The job characteristics (like arrival time, job size, and execution time) are directly read from the archive. Since the supercomputer used may be larger than the one that is being simulated, jobs which cannot fit in the simulated system are left out. Workload flurries [91] are also left out.
2.2.2 Communication within the jobs

We consider communication in all the jobs to be synchronous. Much as the mode of communication is the same, the intensity of communication may vary from job to job. Jobs with high communication intensity, if co-allocated, are expected to have a higher increase in runtime compared to those with low communication intensity.

2.3 The Multi-cluster System

2.3.1 The architecture

We consider a multi-cluster system is made up of $N_c$ homogeneous clusters $C_1, C_2, C_3, \ldots C_{N_c}$. The clusters have the same number of processors ($N_p$) and process jobs by pure space-slicing. The processors in the clusters are connected by identical (local area) links of speed $S$. The clusters are connected together by a backbone of speed $s$. Logically, the clusters are connected by a homogeneous mesh WAN. Co-allocation is possible in the system.

![Figure 2.1: The multi-cluster architecture](image-url)
2.3.2 Queue and scheduler

The system is made up of one queue and one scheduler. All jobs submitted by different users submitted in a single queue and the scheduler picks them from the queue and allocates them to the clusters. A job can be processed in any of the clusters. If the job has multiple components, a component can be processed in any of the clusters. Any cluster can process at most one component of a certain job. Migration is not possible on the system. A scheduler cannot transfer a running job so as to complete execution on another cluster.

2.4 Metrics, Constraints and Objectives

2.4.1 Performance metrics

We use both user based and system based performance metrics. However, our prime interest is to look at performance (and fairness) from the user’s point of view. For user based metrics, we use average waiting time (AWT) and average response time (ART). The two metrics are used interchangeably in this work. This is because they lead to the same conclusion in a non time-slicing case which is considered in this work. For system based metrics, we use the average system utilization (= load at a steady state). These metrics have been highly used in previous related work [26].

2.4.2 The constraints

The main constraint in our work is the availability of processors. The processors in the system are fixed, the jobs are rigid and job migration is not possible. The scheduler therefore has to allocate jobs within the available processors. Communication is an indirect constraint. This is because when a job is co-allocated, it communicates across slower inter-cluster links. This increases the execution time. The job therefore occupies the processors for a longer period of time making them scarcer.
2.4.3 The objectives

In principle, the objective is to minimize average waiting/response time subject to the processor (and communication) constraints. We also aim at maximizing the utilization of the system. However, our objective goes beyond plainly minimizing AWT/ART and maximizing utilization. This is because:

(i) A low AWT/ART may be a result of favoring small jobs and discriminating large jobs. This is unfair and undesirable.

(ii) A high utilization may not necessarily imply a higher rate of processing the actual load. It could be, for example, a result of jobs occupying processors for longer as a result of increased execution time due to co-allocation.

The results, before deducing whether they are more or less desirable, need some intuitive interpretation to make a conclusion on whether or not they are actually desirable in practice.

2.5 Simulation Environment

We use the C++ version of the Mesquite CSIM 18 discrete event simulation engine [85] for our simulations. The engine was used due to its suitability and representation of the different aspects of our set up. Where we used workloads from the traces, we terminated the simulations after running a pre set number of jobs. Where we used synthetic workloads, the simulations were terminated when the results got have a maximum absolute error of 0.08 at 95% confidence interval.

2.6 Results Presentation

We use three approaches to represent our results:

(i) Variation of AWT/ART with utilization:

In this approach, we get the variation of AWT/ART with system uti-
lization. A scheduler (instance), which has a lowest AWT/ART at a certain utilization value, has the best performance. This is used when seeking to get the general comparison in performance of two or more schedulers.

(ii) Variation of AWT/ART with parameter:
In this approach, we plot the variation of the AWT/ART with the parameter of interest. This could be a job stream, system or scheduler parameter. We mostly use this approach when investigating performance sensitivity to the parameter or bounds/ranges in which a certain scheduler outperforms another.

(iii) Using AWT/ART at an instance:
In this approach, we fix all system, job stream and scheduler parameters and find the value of AWT/ART. This is done when we are interested in a comparison of schedulers in a specific fixed circumstance.

The means of result presentation depends on the parameter(s) being investigated and the nature of the deductions sought.
Group-wise Performance Evaluation

Outline: In this chapter, we study the relative performance of jobs grouped by selected parameters. We also study the groups’ performance sensitivity to changes in selected job stream and scheduler parameters. In Section 3.1, we discuss the background/motivation to this work. We describe how large jobs are broken into components (partitioning) and ways groups are generated in Section 3.2. In Section 3.3, we describe the experimental set up of our study. The scheduling algorithm and placement policy used are described in Section 3.4. We study the groups’ relative performance in Section 3.5 and then performance sensitivity to changes in scheduler/job stream parameters in Section 3.6. We study the effect of the partitioning heuristics in Section 3.7 and conclude the chapter in Section 3.8. This chapter is based on [59][60].

3.1 Background

3.1.1 Workload mix and perceived performance

The performance of a computer system not only depends on its design and implementation but also on the workloads it handles [27][35]. Some times, the deductions made are more influenced by the job stream characteristics than the scheduler heuristics [25][26][27].

Basically, schedulers allocate jobs to existing free processors in the system. Depending on the job mix within the job stream, some job streams are harder
to schedule compared to others. The workload can therefore lead to different perceptions on system performance.

Typical computer workloads are skewed. The majority of the jobs are small but they do not constitute the majority of the load [19][56]. Due to differences in their size (or service requirements in general), some jobs are easier to schedule than others. In some scheduling algorithms (like backfilling [57][58] and Fit Processors First Served (FPFS) [2]), jobs can be fished from deep in the queue and get processed. The performance of a certain job, therefore, depends on how it exploits the scheduling opportunities offered by the scheduler. Different jobs, depending on characteristics, have different performances. Changing scheduler parameters can give advantage to some jobs and give disadvantage to others. The trend of the majority of the jobs can be seen as the trend for the entire job stream. This may hide details like discrimination within the job stream which may be important in practice.

Average metric values do not give the correct performance implications if used on skewed job streams [37]. What is shown as an entire job stream improvement in performance could actually be an improvement on a portion of the job stream. Some details may be invisibly hidden. Due to this invisibility, the scheduler cannot be improved to address them. The different portions of the job stream therefore need to be analyzed independently to get a deeper understanding of scheduler performance. The groups have to be carefully demarcated so that jobs are grouped using (approximate) schedulability.

3.1.2 Groups and performance interpretation

Some research work exists on group-wise performance evaluation. Results show that it is of paramount importance in understanding scheduler performance. Srinivasan et al. [76] studied the robustness of schedulers for moldable jobs. They showed that changing scheduler parameters can lead to different performance trends for different job groups (grouped by size). The net change in performance, therefore, does not necessarily represent the trend of all the jobs. A general improvement can actually conceal some cases
of performance deterioration of some jobs in the job stream (and vice versa). Srinivansan et al. [77] further showed that the actual differences in conservative and aggressive backfilling depend on the job mix. They observed that both conservative and aggressive backfilling give performance advantage/disadvantage to specific jobs depending on the number of processors a job requires and the execution time. Selectively making reservations for jobs that have shown evidence of substantial discrimination leads to overall performance improvement [78]. Using job groups, Feitelson [35] showed that it is the interaction between the metrics and workloads, rather than the outcomes of the performance evaluations that reflects the performance characteristics of the studied system.

If we are to get detailed understanding of intrinsic features of parallel job scheduling, there is a need to study group-wise performance in more detail. Most previous studies in group-wise evaluation considered a single parameter for classification. Less study has been carried out on group-wise sensitivity to scheduler and job stream parameters. We use three classification parameters to study (relative) group-wise performance and sensitivity to scheduler and job stream parameters. The parameters used are job size, number of components and width of the widest component.

### 3.2 Partition Heuristics & Group Generation

When preparing job streams for co-allocation, large jobs are broken into components (partitioned). Largeness of a job is relative to other jobs in the job stream. It is also relative to the size on the system in which the job stream is going to be scheduled. For example, if each cluster has 20 processors, a job of size 21 is too big to be processed without being broken into at least two parts (and be co-allocated). However, for a job of size 19, it is optional to have it co-allocated. However, it may be beneficial to co-allocate it. Only jobs considered big are broken into components. We consider a parameter $thres$ which is in such a way that all jobs whose size is less than $thres$ are considered small and therefore not broken into components. Jobs whose size
is greater than or equal to \( \text{thres} \) are broken into components.

If a job of size \( s \geq \text{thres} \) is to be broken into \( c \) components, then the first \( c - 1 \) components are made to have width equal to \( \lfloor \frac{s}{c} \rfloor \) each. The \( c^{th} \) component is made to have a width equal to \( s - (c - 1)\lfloor \frac{s}{c} \rfloor \). This way, we ensure that the components are close to each other in width.

3.2.1 Partition heuristics

We use the random and phased approach to job partition.

The random approach

In the random approach, the number of components a job should have is randomly assigned to all jobs where \( \text{size} > \text{thres} \). If, for example, jobs can be broken into 2, 3, \ldots \( k \) components, every job (whose \( \text{size} \geq \text{thres} \)) has a probability of \( \frac{1}{k-1} \) of being broken into 2, 3, \ldots or \( k \) components. This approach represents a situation where owners determine the number of components for their jobs.

The phased approach

In the phased approach, the number of components a job is broken into depends on its size. If large jobs are to be broken into 2, 3, \ldots \( k \) components, then we make \( k - 1 \) approximately equal portions of the large jobs. Jobs in the smallest portion are broken into 2 components each, those in the second portion are broken into 3 components, and so on. Jobs in the \( k - 1^{th} \) portion are broken into \( k \) components. This approach represents the situation where the system, using size, determines the number of components a job should be broken into.

3.2.2 Groups generation

To generate groups, we use the number of components, the size of the job and the width of the widest component.
Grouping by number components

In this approach, we classify the jobs by the number of components they have. If the maximum number of components a job can be broken into is $k$, then we generate groups $C_1, C_2, \ldots C_k$. Jobs with one component are in group $C_1$, those with two components are in $C_2$ and so on up to when we reach $C_k$. Jobs in $C_1$ are those whose size is less than $\text{thres}$ and therefore are not broken into multiple components.

Grouping by size

In this approach, we use the size of the job to determine the group in which it belongs. If we make $k$ groups, we generate groups $S_1, S_2, \ldots S_k$. Group $S_1$ is made up of the smallest jobs making a proportion of $\frac{1}{k}$ of the jobs while $S_k$ is made up of the largest proportion $\frac{1}{k}$ of the jobs.

Grouping by width of the widest component

This grouping is done like in size grouping but we use the width of the widest component or size for one component jobs. If we are to group the job stream in $k$ groups, we generate groups $W_1, W_2, \ldots W_k$. Group $W_1$ constitutes of jobs with the smallest widest component while $W_k$ has the widest components. Single component jobs are considered to have the widest component width equal to the job size.

3.3 System and Job Stream Instances

We consider a system of 5 clusters of 20 processors each. The job stream is generated from $D(0.85)$ on the interval $[1, 38]$. The mean inter-arrival time is 0.64. This generates a job stream of average size 5.03 and load 0.786. We consider this load to be high enough to bring out the scheduler effect on different job groups.

A job can have a maximum of four components (hence four component based groups). We also make four size based and four widest component
based groups. For the earlier part, we set $thres = 11$ and use random partitioning. The effect of $thres$ and partition approach are studied in Section 3.6.2 and Section 3.7 respectively.

Table 3.1: Boundaries and composition of size and widest component based groups

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Size</th>
<th>Widest Component</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group</td>
<td>Range</td>
</tr>
<tr>
<td></td>
<td>$S_1$</td>
<td>1−1</td>
</tr>
<tr>
<td></td>
<td>$S_2$</td>
<td>2−3</td>
</tr>
<tr>
<td></td>
<td>$S_3$</td>
<td>4−7</td>
</tr>
<tr>
<td></td>
<td>$S_4$</td>
<td>8−38</td>
</tr>
</tbody>
</table>

In Table 3.1, we summarize the percentage compositions and boundaries of the size and widest component based groups. The percentage compositions are computed for the number of jobs and the load. The boundaries are in such a way that the groups have approximately the same number of jobs. The boundaries are therefore the corresponding size/widest component lower quartile, median and upper quartile. For components based groups, the proportion of $C_1$ jobs depends on the value of $thres$. For $thres = 11$, $C_1$ jobs constitute 89.6% of the jobs. This implies that $C_2$, $C_3$, and $C_4$ constitute 3.5% of the jobs each.
3.4 Scheduling Algorithm and Placement Policy

3.4.1 Scheduling algorithm

We use the Fit Processors First Served (FPFS)[2] scheduling algorithm. In FPFS, jobs are queued in their arrival order. When searching for the next job to process, the scheduler starts from the head of the queue and searches deeper into the queue for the first job that fits into the system. In case one is found, it jumps all jobs ahead of it in the queue, gets allocated to the clusters and starts execution. If none is found, the scheduler waits for a job to finish execution or a job to arrive and the search is done again. There is a possibility of starvation of some jobs as they are continuously jumped by other jobs from deep inside the queue. This is avoided by limiting the number of times (to \(max.Jumps\)) a job can be jumped while at the head of the queue. After being jumped \(max.Jumps\) times, no other job is allowed to jump a job at the head of the queue (and get allocated to clusters) until enough processors have been freed (by terminating jobs) to have it start processing. We use FPFS\((x)\) to represent FPFS when \(max.Jumps = x\).

3.4.2 Placement policy

To map components to clusters, we use the Worst Fit (WFit) placement policy. In WFit, the \(i^{th}\) widest component is placed in the \(i^{th}\) freest cluster. It tends to balance the load among the clusters as well as leaving the free processors as evenly distributed as possible among the clusters.

3.5 Group-wise Relative Performance

In this section, we study the relative performance of the different job groups at a fixed load/utilization. We also study the performance variation with load/utilization.
3.5.1 Relative performance at fixed load

We now compare the performance of the different groups. We use $FPFS(10)$ and FCFS (equivalent to $FPFS(0)$) scheduler instances. The details on the effect of $maxJumps$ on performance is studied in Section 3.6.1. We summarize our results in Figure 3.1. From Figure 3.1, we observe that:

(i) All job groups perform better when scheduled by FPFS(10) than when scheduled by FCFS;

(ii) There are some small differences in performance of the different job groups when scheduled by FCFS;

(iii) There are big performance differences among the different groups when scheduled by FPFS(10); and
(iv) Generally, big jobs perform poorly.

Over all, we observe a benefit of allowing small jobs to jump large jobs and get scheduled. This leads to performance benefit even for the jobs that are predominantly jumped. However, the jumping jobs get a higher benefit hence resulting in increased performance differences between the different groups.

### 3.5.2 Performance variation with utilization

In Section 3.5.1, we used a fixed instance of mean inter arrival time (hence load). This may not be a general representation in case the relative group performance varies with load. We now study the variation of performance with utilization (load at a steady state) for the different groups. We summarize the trends in Figure 3.2. In Figure 3.2, we observe that the increase in load/utilization leads to an increase in the average response time of the different job groups. Overall, the increase in utilization does not affect the relative performance of the different groups. Groups with small jobs perform better than groups with large jobs at all utilizations. This implies that the schedulability of the jobs is actually independent of the load on which it is being scheduled. This further implies that we can use a single load/utilization point comfortably to represent the other instances. Never the less, the utilization needs to be high enough so as to portray the characteristics of the scheduler. At low utilization/load, all schedulers work like FCFS.

### 3.5.3 Implication of the results

We now discuss the implications of our results in the general perception of performance in parallel job scheduling.

**The role of maxJumps**

The improvement in performance as the maxJumps value increases from 0 (FCFS) to 10 can be explained by the global effect of allowing some jobs to jump others and get scheduled. This can be looked at in the positive and
negative perspective. On the negative side, the jobs that jump may delay the time at which the jumped jobs start execution. This is because unlike in backfilling, job run times are unknown to the scheduler. They are therefore not put into consideration while allowing jobs to jump. On the positive side, the jobs that jump actually execute on processors that would be idle in FCFS. This implies that they minimize fragmentation and increase the rate at which the load is processed. Picking jobs from the queue also makes it shorter. This has two effects:
(i) It reduces the expected waiting time of the jobs initially behind the picked job; and

(ii) It shortens the queue which is to the advantage of jobs yet to arrive into the queue. This is because they find a shorter queue.

This collaborates with some previous studies. Studies by Chiang et al. [18] showed that in aggressive backfilling, it is rare for a backfilled job to delay jobs deep into the queue. In our case however, the effect can go deeper than that studied by Chiang et al. [18]. This is because aggressive backfilling, unlike FPFS, ensures that the reservation of the job at the head of the queue keeps fixed. Never the less, while the disadvantages of allowing jobs to jump in FPFS affects a few jobs at the head of the queue, the advantages go beyond the queue (to jobs yet to arrive). By the moment the job reaches near the head of the queue, it has accumulated benefits in performance beyond the possible set back it can get by being delayed by jobs that jump it. The advantages outweigh the disadvantages leading to a net gain to all jobs.

Job characteristics and schedulability

We observe that for both FCFS and FPFS(10), different jobs groups perform differently. This implies that the different job characteristics have an effect on how easy/hard a job is to schedule.

Since FCFS does not allow jobs to jump others, a job at the head of the queue blocks all others behind it until there are enough free processors for it to start execution. There is therefore little performance difference. The slightly better performance of small jobs can be attributed to the fact that they are less likely to wait for long time when they reach the head of the queue. Since FPFS allows mostly small jobs to jump and get scheduled before others ahead of it, there is good performance for small jobs and a relatively poor performance for large ones.

We observe a direct relationship between (i) performance and size and (ii) performance and width of the widest component. This direct relationship is missing for number of component. However, the worst performance of jobs
in $C_2$ shows that it is likely to be due to the width of the widest components (since job sizes are equally distributed). This shows that the width of the widest component has a big effect on job schedulability. Its effect is caused by the way free processors are distributed in the clusters as dictated by the placement policy. Since WFit places the widest component in the freest cluster, it distributes the free processors among the clusters as evenly as possible. It is therefore hard to get a big block of free processors in a single cluster to process a wide component. Since FPFS allows (small) jobs that can fit to jump those which cannot fit and the placement policy places it in the freest cluster, jobs with wide components are disadvantaged more.

Relative schedulability among the groups

Our results also show that the relative schedulability among the job groups is independent of the load and does not vary linearly with the characteristics. Jobs in $S_1$ and $S_2$ for example, have a closer performance and those in $S_4$ perform un-proportionately poor. The same trend is observed in jobs grouped by the number of components and width of the widest components. The extent of relative schedulability among the jobs also depends on the scheduler used. We observe less relative schedulability in jobs scheduled by FCFS compared to those scheduled by $FPFS(10)$.

Schedulability and job stream performance

Some of the characteristics that affect schedulability can be adjusted by the system/user while others cannot. The factors that directly affect schedulability are size and width of the widest component. The size cannot be adjusted by the system for rigid jobs but the width of the widest component can. It is therefore possible to improve schedulability by appropriately partitioning the jobs. This should, in principle, aim at minimizing the width of the widest component. This, together with the scheduling algorithm, can lead to improved performance. A more detailed study of the effect of the partition heuristics is done in Section 3.7.
3.6 Performance Sensitivity to Parameters

In Section 3.5 we have observed that there are differences in schedulability (and hence performance) of the different job groups. This has been done on fixed scheduler parameter (\(\text{maxJumps}\)) and fixed job stream parameter (\(\text{thres}\)). We now study the sensitivity of the performance (and relative performance) of the different groups with the two parameters.

### 3.6.1 Sensitivity to maxJumps

We vary \(\text{maxJumps}\) from 0 to 50 and summarize the performance trend of size, components and widest component based groups in Figure 3.3. From Figure 3.3, we observe that:

(i) Increasing the value of \(\text{maxJumps}\) leads to an improvement in performance in all the job groups;

(ii) When \(\text{maxJumps}\) value is low, increasing it leads to more performance benefits compared to a case when \(\text{maxJumps}\) is high;

(iii) There is a \(\text{maxJumps}\) value beyond which a further increment leads to minimal benefit in performance. This \(\text{maxJumps}\) value is higher for component groups than for size and widest component groups; and

(iv) Groups with large jobs perform poorer compared to groups with small jobs at all \(\text{maxJumps}\) values.

Overall, we observe an improvement in performance as \(\text{maxJumps}\) is increased. All jobs register improvements in performance much as large jobs perform relatively poorer compared to small jobs.

### 3.6.2 The effect of thres

We now study the effect of \(\text{thres}\) on the (relative) group-wise performance. We vary \(\text{thres}\) from 3 to 19 and summarize our results in Figure 3.4. We use only size-based job groups since all groups take a similar trend. From Figure
3.4 we observe that:

(i) Increasing $thres$ leads to poorer performance for all job groups and

(ii) There is a higher rate of change for large jobs compared to small jobs

Overall, we observe that a lower value of $thres$ yields better performance for all the jobs. Increasing $thres$ leads to poorer performance but large jobs get a higher rate of performance deterioration.
3.6. PERFORMANCE SENSITIVITY TO PARAMETERS

3.6.3 Implications of the results

The effect of maxJumps can be attributed to the effect of the scheduling opportunities given to the jobs as the scheduler allows them to jump. Small jobs utilize the free processors that give them good performance. In so doing, they minimize cases of fragmentation which in turn improve the performance of the large jobs. Since small jobs constitute a very small proportion of the load, they can hardly fill the small spaces caused by fragmentation. At very high maxJumps, the small jobs get processed immediately they arrive. We therefore observe that at very high maxJumps, small jobs have the best possible performance (ART = mean execution time = 10).

The effect of thres is caused by the packing benefits of breaking up the jobs. If thres is low, a big proportion of the jobs is broken into components. This reduces the average width of the widest component in the job stream which improves job schedulability within the job stream. If thres is high, then jobs which are not broken into components tend to be hard to schedule. They are jumped by smaller jobs. This leads to better performance of small jobs. However, if the large job is jumped maxJumps times, all the jobs behind it are held back until it is scheduled. This leads to an overall poor performance but a relatively better performance of small jobs. This shows

Figure 3.4: Performance of job groups with thres for FPFS(10)
us that co-allocation is of a performance benefit to the entire job stream. We however note that the benefits of co-allocation are partially countered by the effect of communication which is not studied in this chapter. The effect of communication is studied in Chapter 5.

3.7 The Effect of the Partitioning Heuristics

3.7.1 Random vs phased approach

We use the phased and random approach and summarize the variation of performance with $\text{maxJumps}$ in Figure 3.5. We use only size based partitions. This is because the group a job belongs to, in size based partitions, is independent of the partition heuristics. This is not the case for components generated by the number of components or the width of the widest component. We only show $S_1$ and $S_4$ trends (other groups show similar trend). From Figure 3.5, we observe that:

![Figure 3.5: Performance sensitivity to maxJumps for random and phased partitioned job stream](image)

(i) Phased partition performs better than random partition;

(ii) Partitioning affects both co-allocated and non co-allocated jobs;
3.8. CONCLUSION

(iii) The value of \textit{max.Jumps}, beyond which a further increase yields negligible benefits, is lower for phased partition than random partition;

(iv) There is a lower performance deviation among groups for phased partition compared to randomly partitioned groups; and

(v) The benefit in performance is also felt by FPFS(0).

We observe that there is a better performance when the job stream is partitioned by the phased approach compared to when it is partitioned by the random approach.

3.7.2 Implications of the results

The improvement in performance due to the different partition approaches is attributed to the ease in packing of jobs with small components. Since phased approach breaks large jobs into more components, the average widest component for the partitioned jobs is lower compared to a case of random partitioning. This implies that the way partitions are made can be of great importance to the performance of co-allocation.

The differences in performance for FPFS(0) implies that the improvements are not necessarily scheduler specific. The reduction in the performance deviation between \( S_1 \) and \( S_4 \) jobs implies that there is less relative starvation among the small and large jobs. This implies that the improvement in performance on the phased approach of partitioning also comes with an improvement in fairness. More studies in fairness are carried out in Chapter 6 and Chapter 7.

3.8 Conclusion

We have studied the performance of processor co-allocation using job groups. We have studied the effect of the jobs characteristics (size, components and width of the widest component) on their schedulability. We have also studied the relative performance of the different groups and their sensitivity to
variations in selected job stream and scheduler parameters. We have also studied the effect of the partition heuristics.

We have shown that jobs characteristics have a big effect on their schedulability and hence performance. Due to large variations in characteristics, different jobs have different performances. Much as the groups have similar performance trends with changes in parameters, the rates of performance changes are different. In cases of performance improvement (like increasing \textit{max.Jumps}), big jobs get comparatively lower rates of improvement. In cases of performance deterioration (like increasing \textit{thres}), big jobs have a higher rate of deterioration. The numerical majority of small jobs make them influence the job stream average metric value. The poor performance of large jobs, that constitute the majority of the load, is not sufficiently implied by the job stream average metric. The partition heuristics have a big effect on the performance of the jobs. Using the phased approach leads to big gains in performance for all the jobs. There is also a smaller difference in performance for a job stream partitioned using the phased approach.

Our results have practical significance. First, the job stream performance is not representative of the performance of large (resource intensive) jobs. Supercomputers are developed to handle resource intensive jobs. Using the job stream average metric values give deductions which are non-representative of the target jobs of the supercomputer. Secondly, a change in job stream performance due to a change in a parameter is an aggregation of the individual changes which are not the same for all the jobs. Large jobs get small improvements but big deteriorations in performance when parameters are changed. To be sure, changes in performance need deeper group level analysis. Finally, partition heuristics give better performance benefits to all jobs in the job stream compared to change in scheduler parameters. Good partitioning is therefore a good option to improve performance of co-allocation. Using both partitioning with parameter variation gives better performance.
The Greedy Multi-cluster Scheduler

Outline: In Chapter 3, we observed large differences in performance between large and small jobs. This implies that different users get different levels of satisfaction. In this chapter, we propose a new algorithm that seeks to reduce the performance gap between the best and worst performing jobs. It also seeks to improve the overall job stream performance. We describe the underlying principle and the approach of our scheduler in Section 4.1. We describe the scheduler in Section 4.2 and the experimental set up to evaluate it in Section 4.3. We compare selected instances of our scheduler with FPFS in Section 4.4 and study its performance sensitivity to parameters in Section 4.5. Finally, we discuss our findings in Section 4.6 and conclude the chapter in Section 4.7.

This chapter is based on the work in [61][62].

4.1 Introduction

4.1.1 Motivation

In Chapter 3, we observed that there are big differences between the performance of small and large jobs when scheduled by FPFS. Small jobs perform excessively well while large jobs perform poorly. This leads to two undesirable scenarios:

(i) The resource intensive/large jobs, which are the prime reasons for using high performance computers, achieve poor performance. This implies
that the target audience for high performance computers actually gets poor service from them;

\( (ii) \) The high differences in performance between large and small jobs lead to differences in levels of satisfaction among users. The system is therefore unfair.

Studies on characteristics of multi-cluster job streams [56] show that jobs vary in characteristics like size, duration and memory. These characteristics define the amount of resources required from the system for the jobs to process. They also determine the job schedulability [59]. Since jobs are not equally as schedulable and some arrive in peak hours while others arrive in off peak hours, they are bound to have differences in performance. However, we note the facts that:

\( (i) \) The principle aim of high performance computers is to process resource intensive (large) jobs. Resource intensive jobs, however, mostly have poor performance due to schedulability constraints;

\( (ii) \) Small jobs, due to their size, are likely to get more scheduling opportunities. They are therefore likely to have good performance. The good performance, however, should not be at the expense of large jobs; and

\( (iii) \) The performance gap between large and small jobs need to be minimized. But this is not to be achieved by merely imposing a performance disadvantage to the small jobs. Instead, it should be by allowing large jobs exploit available scheduling opportunities and ensure that the good performance of small jobs is not at the expense of large jobs.

There is, therefore, a need to give an (artificial) advantage to naturally disadvantaged jobs so that their performance is improved as much as possible. This has to be done in such a way that the schedulable jobs are only prevented from delaying the unschedulable jobs but not from taking advantage of their schedulability. When evaluating/comparing schedulers, we need to look at the group-wise performance together with that of the entire job stream. This
4.1. INTRODUCTION

will help understand whether the overall good performance of a scheduler is due to starvation of resource intensive jobs.

4.1.2 Approach

Our approach bases on the generic greedy algorithm. We briefly describe the greedy algorithm and then discuss how we modify it to suit the parallel job scheduling set up.

The generalized greedy algorithm

A greedy algorithm is an algorithm that ensures that it makes the locally optimum choice at any stage with a hope of ending up at the global optimum [20]. It has been used to solve generic problems like the knapsack problem and the traveling sales man problem. Generally, in the greedy approach:

(i) We define the objective; and

(ii) We ensure that if at iteration $k$ there are $n_k$ options (possible steps), we choose the option that moves as close to the objective as possible (local optimal step).

In the 0/1 knapsack problem for example, we have to fill (indivisible) items in a knapsack of fixed volume. Each item has a volume and a value. Our objective is to maximize the value in the knapsack. At any iteration (adding an item in the knapsack), we compute the value per volume of each remaining item and the item with the highest value per volume that can fit in the remaining knapsack volume (local optimum), is chosen.

We use this concept of making the (approximate) most optimal step possible when scheduling jobs on multi-cluster systems. However, we make some modifications to it in order to suit the multi-cluster set up.

Modification to the parallel job scheduling case

The modifications in the greedy algorithm (to generate a greedy scheduler) put into consideration the operational differences of the parallel job schedul-
ing environment and a typical environment (like knapsack case) where the greedy algorithm has been used. The main differences are:

(i) Selectivity:
The general greedy algorithm is selective. It picks some of the items and leaves the others. Parallel job scheduling is exhaustive. The scheduler has to schedule all the jobs. While the general greedy algorithm is concerned with 'what', the scheduler has to be concerned with 'what' and 'when'.

(ii) Direct and indirect convergence to the optimal
The general greedy scheduler only approaches the optimal value directly i.e. by considering the next item to pack so as to tend towards the optimal. However, the scheduler has to consider the job it chooses next, how the job chosen affects the objective as well as the way the jobs not chosen will affect the objective when (not if) they are eventually chosen.

If the greedy algorithm was to be directly transformed into the greedy scheduler, it would ensure that the next job to schedule (directly) optimizes the objective. Therefore, the latest job to arrive would have the highest priority (Last In, First Out). This definitely starves jobs and is therefore unfair. In our approach, we aim at optimizing \( AWT \) by ensuring that:

(i) A job that will give a big negative impact on the objective if unscheduled, is given a high priority; and

(ii) If we have two jobs \( J_x \) and \( J_y \) where it is easier to schedule \( J_y \) after scheduling \( J_x \) than it is to schedule \( J_x \) after scheduling \( J_y \), then \( J_y \) is given higher priority.

In (i), the scheduler is preemptive. A job is scheduled so as to preempt it from worsening the objective when scheduled later. This gives a FCFS priority to the jobs. In (ii), the scheduler is optimistic. A job is scheduled when the scheduler is optimistic of achieving the objective faster because the unscheduled jobs will not negatively affect it. This gives jobs priority according
to their schedulability. Schedulable jobs have lower priority compared to unschedulable ones.

The two factors are not always in agreement. We therefore allow both of them to contribute to the prioritization criteria. The strongest factor on any job dominates its priority. For example, in case the time spent in the queue (seniority) is nearly the same but there is a big difference in schedulability, then schedulability overrides seniority. The same case applies in case of big differences in seniority. To illustrate the performance benefit of our approach, we use the example below.

**Example 4.1.1** Let us consider two jobs $J_1$ and $J_2$ of size 11 and 4 respectively. $J_1$ is broken into two components of width 6 and 5 and is to be co-allocated. $J_1$ arrived $\delta t$ units of time after $J_2$. They are to be scheduled on a 4 cluster system. At the instance of scheduling, clusters $C_1$, $C_2$, $C_3$ and $C_4$ have 5, 8, 6 and 1 free processors respectively. Let us consider two cases:

**Case - I:** where jobs are only prioritized by the time spent in the queue and

**Case - II:** where the job size also contributes to prioritization in Case I.

Let us examine how the schedules are in each of the cases.

**Case I** If the priority is based on the arrival order, then $J_2$ will be scheduled before $J_1$. $J_2$ will be processed on $C_2$ leaving it with 4 free processors. $J_1$ cannot start processing unless some of the running jobs terminate.

**Case II** Since $J_1$ is bigger than $J_2$, its priority will be increased because of its size. If after the increment its priority is still lower than $J_2$, then Case I above is still valid. If $J_1$ gets a higher priority than $J_2$, then it will be scheduled first. Its components will be processed in cluster 2 and 3 leaving them with 2 and 1 free processors respectively. $J_2$ will also be processed in $C_1$ leaving it with 1 free processor.

From example 4.1.1, we observe that giving large jobs a priority advantage can lead to better packing, higher system utilization, and hence better performance (size in this case approximates schedulability). This is because the small jobs have more chances of being processed. Giving small jobs earlier
processing opportunities can block large jobs. This leads to system fragment-
tation and poor performance of large jobs.

We therefore use a combination of schedulability and seniority to prioritize
the jobs so that we improve job packing, overall performance and minimize
the discrimination large jobs face.

4.2 The Scheduler

4.2.1 Background

The main idea behind the greedy scheduler is to give an artificial priority
advantage to naturally disadvantaged (large) jobs. However, the scheduler
can try a lower priority job in case the high priority job can not fit in the
available free processors. This improves the packing scheme which improves
the performance of large/un-schedulable jobs without highly affecting that
of small jobs. We separate priority from the queuing policy. While a job
remains in the queuing order, they are searched for in the order of priority
different from the queuing order. As a job spends more time in the queue,
the rate of priority change depends on its (estimated) schedulability. This
implies that the relative priority between any two jobs, over the time they
spend in the queue, may change. Increasing the priority of jobs that are hard
to schedule gives them more scheduling opportunities.

Our studies in Chapter 3 show that allowing lower priority jobs to jump
higher priority jobs that do not fit in the system gives performance benefits
to both the jumping and jumped jobs. This implies that jobs do not have
to be processed in the strict priority order so as to get good performance. If
the highest priority job cannot fit in the available processor hole, the next
job needs to be considered.

However, we assume that the users cannot accurately estimate the run-
time of their jobs [54]. This implies that job runtimes are unknown to the
Schedulers. Continuously allowing low priority jobs to jump high priority
jobs can cause starvation. Starvation, in our case, cannot be controlled like
in (conservative/aggressive) backfilling [57][58]. There is therefore a need to protect the job at the head of the queue from possible (excessive) starvation. This is done, like in FPFS [2], by limiting the number of times (to \textit{maxJumps}) a job at the head of the queue can be jumped. After being jumped \textit{maxJumps} times, all attempts to have other jobs scheduled are halted until enough space has been created (by terminating jobs). The job at the head of the queue is then scheduled.

### 4.2.2 Priority estimation

Let us consider a job \( J_i \) that arrived at time \( t_{ia} \). At an arbitrary time \( t \), \( J_i \) has spent \( t - t_{ia} \) units of time in the queue. Job schedulability is approximated by its \textit{hardness function} \( h \). Job hardness is considered to be a function of the number of components \( n \), the width of the widest component \( w^* \) and size \( w \) of the job [59][60]. The hardness function \( h(w, n, w^*) \) increases with each of the parameters. In this work, we consider a linear function

\[
h(w, n, w^*) = \alpha w + \beta n + \gamma w^* \quad (4.1)
\]

where \( \alpha \), \( \beta \), and \( \gamma \) are positive constants. The priority indicator \( I_i(t) \) of \( J_i \) at time \( t \) is obtained by multiplying the time it has spent in the queue by its hardness value.

\[
I_i(t) = (\alpha w_i + \beta n_i + \gamma w_i^*) \times (t - t_{ia}) \quad (4.2)
\]

where \( w_i \), \( n_i \) and \( w_i^* \) are the \( w \), \( n \) and \( w^* \) values for \( J_i \) respectively. The \( I_i(t) \) values are computed at every scheduling attempt. We set \( \alpha = \beta = \gamma = 1 \) in our initial studies. The effect of the (relative) values of \( \alpha \), \( \beta \) and \( \gamma \) are studied in Section 4.5.3.

### 4.2.3 Job selection

When searching for the next job to process, the jobs are searched in reducing order of \( I_i(t) \). To minimize the resources spent on continuously computing \( I_i(t) \), the computation of \( I_i(t) \) (and hence the search for the next job to be scheduled) is done up to a certain number of jobs (\textit{depth}) in the queue.
Limiting candidate jobs to depth also reduces cases where jobs are fished from deep inside the queue. If jobs are allowed to jump from deep inside the queue, we create opportunities for system fragmentation which is disadvantageous to the large jobs closer to the head of the queue.

### 4.2.4 The algorithm

We now describe the step by step flow of the greedy multi-cluster scheduler.

1. Check the times a job at the head of the queue has been jumped
   1.1 If it is jumped less than \( maxJumps \), go to step 2
   1.2 If it is jumped \( maxJumps \) times, check if it can fit in the system.
   1.3 If it can fit in the system, start its execution and go back to 1
   1.4 If it cannot fit, wait until enough space is created in the system, start its execution and go back to 1.

2. Compute the priority indicators for the first \( depth \) jobs

3. In reducing order of indicators, check for the first job that fits in the system.
   3.1 If a job is found, schedule it
   3.2 If none fits, wait until a job finishes execution and repeat step 2

4. If the job scheduled was not from the head of the queue, increment the number of times the job at the head of the queue is jumped by 1

5. Repeat the process starting from 1 until all jobs are finished.

As a notation, we use Greedy \((j, d)\) to represent the greedy scheduler when \( maxJumps = j \) and \( depth = d \)
4.3 Experimental Set Up

We now describe the experimental set up to study the performance characteristics of the greedy multi-cluster scheduler.

4.3.1 The multi-cluster system

We consider a system of five homogeneous clusters of 20 processors each. The clusters process by pure space slicing. The clusters are connected by fast dedicated wide-area links of negligible communication latency. Co-allocation is possible on the system. The system is served by one queue and one scheduler.

4.3.2 The job stream

We use a synthetic workload. We generate the workload from $D(0.85)$ over the interval $[1, 38]$. We consider exponentially distributed inter-arrival and execution times. The mean execution time is 10.0. We use $\text{thres} = 11$, this implies that the largest 10% of the jobs are co-allocated. We use the random approach to break up all jobs whose size is greater than $\text{thres}$ into 2, 3 or 4 components.

For performance evaluation, we use four size based groups $S_1, S_2, S_3$ and $S_4$. Their boundaries and proportions in the job stream are summarized in Table 4.1. We use only size based groups because the group of a job is independent of the value of $\text{thres}$ and partition heuristics.

Table 4.1: Job group boundaries and numeric/load contributions

<table>
<thead>
<tr>
<th>Group</th>
<th>Size range</th>
<th>Jobs (%)</th>
<th>Load (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>1 – 1</td>
<td>24.88</td>
<td>5.08</td>
</tr>
<tr>
<td>$S_2$</td>
<td>2 – 3</td>
<td>25.64</td>
<td>11.49</td>
</tr>
<tr>
<td>$S_3$</td>
<td>4 – 7</td>
<td>25.50</td>
<td>24.13</td>
</tr>
<tr>
<td>$S_4$</td>
<td>8 – 38</td>
<td>24.28</td>
<td>59.50</td>
</tr>
</tbody>
</table>
CHAPTER 4. THE GREEDY MULTI-CLUSTER SCHEDULER

4.4 Performance Comparison with FPFS

First, we compare the performance of selected instances of the greedy multi-cluster scheduler with selected instances of FPFS scheduler. We fix the value of \( \text{maxJumps} \) (for both FPFS and the greedy scheduler) to 5 and use two \( \text{depth} \) values of 5 and 20 for the greedy scheduler. A detailed study on the effect of \( \text{maxJumps} \) and \( \text{depth} \) on scheduler performance is done in Section 4.5. We make comparisons for the entire job stream as well as for size based groups.

4.4.1 Relative performance for the entire job stream

![Graph showing relative performance](image)

Figure 4.1: Relative performance of Greedy(5,5), Greedy(5,20) and FPFS(5)

Figure 4.1 shows the relative performance of Greedy(5,5), Greedy(5,20) and FPFS(5) for the entire job stream. We observe that at high utilization, Greedy(5,20) performs best while Greedy(5,5) performs worst. This implies that at fixed \( \text{maxJumps} \), a low value of \( \text{depth} \) leads to poor performance of the greedy scheduler. FPFS therefore outperforms the greedy scheduler. However, increasing \( \text{depth} \) leads to improvement in the performance of the greedy scheduler that it outperforms FPFS.
4.4. PERFORMANCE COMPARISON WITH FPFS

4.4.2 Relative performance for job groups

Figure 4.2 shows the relative performance of Greedy(5,5), Greedy(5,20) and FPFS(5) for size based groups $S_1$, $S_2$, $S_3$ and $S_4$. We observe that the different groups do not follow the relative performance trend of the entire job stream. For $S_1$ and $S_2$, we observe that FPFS(5) performs best while Greedy(5,5) performs worst. For $S_3$ Greedy(5,20) performs best while Greedy(5,5) and FPFS(5) perform approximately equally. For $S_4$, Greedy(5,20) performs best and FPFS(5) performs worst.
4.4.3 Explanation for the relative performance

We observe that the relative performance of FPFS and the greedy scheduler are different for different groups. The relative performance trends can be explained by the intrinsic pros and cons of the approaches to scheduling. These are the packing scheme and the restrictive effect of the $depth$.

The packing scheme

How good a packing scheme is depends on how well it utilizes the available processors in the system. If the processors are well utilized, then all the competing jobs benefit. While FPFS tries to schedule jobs in the order of their arrival, the greedy scheduler enhances the FPFS priority by the job schedulability. The greedy scheduler gives a high priority to a large job compared to a small job that arrived slightly earlier than it. It is easier to schedule a small job after a large job than vice versa. This is because the small job can easily fit in the residual free processors in the system and the large job can easily become a victim of system fragmentation.

Workload studies show that the majority of the jobs in a typical supercomputer job stream are small [19][56]. This implies that the majority of the jobs processing at a certain time are small. In cases where a system cannot accommodate any other job at a certain time, a small waiting job will require fewer jobs to terminate so as to get scheduled. This implies that giving a scheduling advantage (higher priority) to large jobs improves their performance without causing a substantial disadvantage to the small jobs.

The restrictive effect of $depth$:

The greedy scheduler picks from the first $depth$ jobs in the queue. This implies that even if there are some free processors in the system, a job beyond $depth$ cannot be allocated to them. A low $depth$ therefore causes high capacity loss in the system. Low $depth$ gives a performance disadvantage to the small jobs that would, if scheduled by FPFS for example, jump and get scheduled. However, since the runtimes of these jobs are unknown, allowing
many small jobs to jump and get processed can lead to system fragmentation. Blocking them reduces fragmentation to the benefit of large jobs with in depth. For good performance, depth needs to be high enough to allow enough jobs to exploit existing processor holes.

Overall, the greedy scheduler benefits from a better packing scheme but suffers from the restrictive nature of depth. The depth therefore needs to be high enough so as to get substantial benefits from the packing scheme.

4.5 Scheduler Sensitivity to Parameters

Our studies in Section 4.4 have considered parametric instances of the greedy and FPFS schedulers. We have observed that the parameters have an effect on the (relative) performance of the schedulers. Adjusting the parameter values can therefore lead to an improvement or deterioration in performance of the scheduler. We now make a deeper study on the effect of the parameter values on the performance of the schedulers. We set mean inter-arrival time to be 0.62 (this generates a load of 0.811). We study performance variation with \textit{maxJumps}, \textit{thres} and the coefficients of the hardness function (\(\alpha\), \(\beta\), and \(\gamma\)).

4.5.1 Performance variation with \textit{maxJumps}

To investigate the effect of \textit{maxJumps}, we fix \textit{depth} to 5 and 20 and study the performance variation with \textit{maxJumps} for both the greedy scheduler and FPFS. We study the trend of the entire job stream as well as size based groups. Figure 4.3 shows the performance variation of the greedy scheduler and FPFS with \textit{maxJumps} for the entire job stream. We observe that:

(i) Increasing \textit{maxJumps} leads to an improvement in the performance for both FPFS and greedy schedulers;

(ii) When \textit{maxJumps} is low (< 20), increasing it leads to substantial improvements in performance;
(iii) For high \( \text{max}\text{Jumps} \) values (\( > 20 \)), increasing of \( \text{max}\text{Jumps} \) further leads to negligible improvement in performance;

(iv) There is a small performance difference between FPFS and the greedy scheduler when \( \text{depth} = 20 \); and

(v) The greedy scheduler when \( \text{depth} = 5 \) performs worst.

Overall, the performance of the schedulers highly depend on \( \text{max}\text{Jumps} \). We get good performance when the \( \text{max}\text{Jumps} \) values are high. Therefore, it is beneficial to use the greedy scheduler when \( \text{max}\text{Jumps} \) is high.

In Figure 4.4, we present the performance variations of the schedulers with \( \text{max}\text{Jumps} \) for the size based job groups. We observe that:

(i) The different groups take the same trend as the entire job stream. However, the schedulers’ relative performances are not the same;

(ii) For \( S_1 \) and \( S_2 \), FPFS performs better than both instances of the greedy;

(iii) For \( S_3 \), FPFS and greedy when \( \text{depth} = 20 \) have a close to similar performance and greedy performs worst when \( \text{depth} = 5 \);
(iv) For $S_4$, the greedy scheduler performs best when $depth = 20$; and

(v) The $maxJumps$ value, beyond which an increase does not lead to improvement on performance is low for small jobs and high for large jobs.

The group-wise performance trends show us that the greedy scheduler leads to better performance of the large jobs while it leads to a poorer performance of the small jobs. It further shows us that though the entire job stream trend may show similar performance, there can be differences in performance when looked at from a group wise perspective.
4.5.2 Performance variation with depth

We now investigate the performance variation of the greedy scheduler with depth at fixed maxJumps. We set the value of maxJumps to 5 and 20 for both the greedy scheduler and FPFS. We then vary depth for the greedy scheduler and study the performance variation. We investigate the entire job stream as well as the size-based groups. We present the performance trend for the entire job stream in Figure 4.5. In Figure 4.5, we observe that:

Figure 4.5: Performance variation for FPFS and the greedy scheduler (maxJumps = 5 and maxJumps = 20) with depth for the entire job stream

(i) An increase in depth leads to an improvement in performance;

(ii) There exists a threshold value of depth beyond which the greedy scheduler outperforms FPFS;

(iii) The threshold depth value is low for low maxJumps values and high for high maxJumps values; and

(iv) When maxJumps is high, there is a smaller benefit of using the greedy scheduler other than FPFS compared to a case when maxJumps is low.
4.5. SCHEDULER SENSITIVITY TO PARAMETERS

The performance trends show us that there is better performance of the greedy scheduler at a high depth. We also observe that there is a small benefit of using the greedy scheduler when maxJumps = 20.

Figure 4.6: Performance variations of FPFS and greedy scheduler with depth for $S_1$ (top left), $S_2$ (top right), $S_3$ (bottom left) and $S_4$ (bottom right)

Figure 4.6 shows the performance variation with depth for the job groups. We observe that:

(i) The performance trend of the job groups is the same for the entire job stream;

(ii) The rate of performance improvement is high for small jobs and low for large jobs;
(iii) For the same job group, there is a high rate of performance improvement at low depth; and

(iv) For large jobs, the greedy scheduler outperforms FPFS and for small jobs, FPFS outperforms the greedy scheduler.

Overall, we observe that the greedy scheduler, like the FPFS scheduler performs better at high \textit{max.Jumps} values.

### 4.5.3 Performance sensitivity to $\alpha$, $\beta$ and $\gamma$

We also studied the effect of the relative values of the hardness function coefficients $\alpha$, $\beta$ and $\gamma$. We used the relative rather than the absolute values of the coefficients since absolute values do not determine the order of the scheduler and therefore have no practical implications. Our studies showed a negligible change in performance as coefficients were varied relative to others. This implies that much as incorporating schedulability estimates brings performance benefits, there is no optimal coefficient combinations. A single parameter, like size can be good enough for the harness function of the greedy scheduler.

### 4.6 Discussion

We have proposed a new multi-cluster scheduler that incorporates greedy techniques in prioritization. We have studied its sensitivity to its parameters as well as studying its performance comparison with FPFS.

Our studies using the entire job stream and job groups based further demonstrated the need for group-wise evaluation as a means of getting to understand scheduler performance in depth. For the entire job stream, Greedy ($\text{depth} = 20$) and FPFS (Figure 4.1) have approximately the same performance. However, when we look at the performance of the different job groups (Figure 4.2), we observe that actually there are bigger differences in performance at job group level. The similarity in job stream average actually does not imply similarities in the levels of satisfaction for the system users.
4.6. DISCUSSION

We have also observed that mostly FPFS outperforms the greedy scheduler for small jobs and the greedy scheduler outperforms FPFS for large jobs. This implies that there is a smaller difference in performance between small jobs and large jobs for the greedy scheduler compared to FPFS. It can also be deduced that owing to the prioritization/packing scheme of greedy scheduler, the performance of the large jobs is improved by minimizing starvation imposed on them by the small jobs. This implies that the users for the greedy scheduler get closer levels of satisfaction compared to FPFS. The greedy scheduler is therefore fairer than FPFS. More work on fairness and its evaluation as well as comparing FPFS and greedy schedulers for fairness is covered in Chapter 6 and Chapter 7.

When we compare the magnitudes of the performance gaps for the different job groups, we observe that FPFS outperforms the greedy scheduler (on small jobs) by a bigger gap when compared to the performance gap by which the greedy algorithm outperforms FPFS (on large jobs). This can be attributed to the resource requirements of the different jobs. The load (processor hours) in a single big job is equivalent to the load in several small jobs. The many small jobs have a bigger effect on the average than the few large jobs. This implies that their benefit in performance makes a bigger improvement on the performance of all small jobs compared to the effect of the large jobs. Processing a large job also requires a contiguous pool of resources. This makes it more prone to fragmentation. The large job therefore shows a numerically small improvement though the load wise benefit in performance is actually large.

The maxJumps parameter improves performance by allowing jobs to jump and fill the processors that would be idle. Increasing it leads to overall performance benefit. The depth parameter actually restricts processing of jobs from deep in the queue. This restrictive effect stops the jobs beyond depth to utilize the available processors. At the same time, the greedy prioritization strategy to a certain extent allows the job that best utilizes the available to processor hole. This reduces the fragmentation and also increases the processing chances of the large jobs closer to the queue without substan-
CHAPTER 4. THE GREEDY MULTI-CLUSTER SCHEDULER

Initially casting a disadvantage to the smaller (lower priority) jobs. If depth and max.Jumps are high enough, the greedy scheduler employs a packing scheme that best utilizes the available processors.

The coefficients $\alpha$, $\beta$ and $\gamma$ have a negligible effect on performance. This is because much as they change the priority indicator of the jobs, they rarely change the relative priority among the jobs. This implies that the jobs are prioritized in the same order where changes in the coefficients do not affect the priority order. The changes therefore do not translate into a different scheduling order hence performance.

In practice, we can deduce that the greedy scheduler outperforms FPFS so long as max.Jumps and depth are high enough. It is also fairer than FPFS.

4.7 Conclusion

In this chapter, we have proposed a new multi-cluster scheduler. Our scheduler improves FPFS [2] by (i) giving a priority advantage to large/un-schedulable jobs and (ii) limiting the depth into the queue, where a candidate job for scheduling can be obtained. In so doing, our scheduler improves the packing scheme and improves the performance of the large jobs as well as the entire job stream. We study the effect of its parameters. Specifically, we have studied the effect of max.Jumps, depth and the hardness function coefficient. We have observed that we get best performance by having a high depth and max.Jumps. The coefficients of the hardness function have a negligible effect on the scheduler performance. Our scheduler also reduces the performance differences between the small and large jobs. It can therefore be deemed fairer compared to FPFS.
Chapter 5

Communication and Co-allocation

Outline: So far, we have considered cases where co-allocation does not affect job runtime. In practice, since a co-allocated job has to communicate through a slower wide-area link, it executes for a longer time than when processed in a single cluster. In this chapter, we study how communication affects the performance of co-allocation. We discuss the models and effects of communication on co-allocation in Section 5.1. In Section 5.2, we discuss the previous related work on the effect of communication on co-allocation. In Section 5.3, we describe the experimental set up of our study. We present our communication model and study how performance is affected by communication based characteristics in Section 5.4. We discuss our results in Section 5.5 and conclude the chapter in Section 5.6. This chapter is based on work in [64].

5.1 Introduction

5.1.1 Communication models

There are two main communication models considered in literature; the synchronous and asynchronous communication model. In the synchronous model, the job execution is made up of successive communication and processing phases. Any execution phase does not start unless the preceding communication phase has been completed by all the processes. Likewise, a communicating phase does not start unless that all processes have completed
the preceding processing phase. Asynchronous communication, on the other hand, job execution consists of processes that run independently but pairs of processes occasionally communicate with each other. Like in previous work whose communication model approach we extend [7], we consider the synchronous communication model.

5.1.2 The effect of communication

Co-allocation involves multi-site processing. The time taken by a communication message between any two processes depends on whether it is an inter-cluster or intra-cluster message. An inter-cluster message takes longer time compared to an intra-cluster message. This is due to a relatively slower inter-cluster link.

Let us consider a multi-cluster system made up of homogeneous clusters joined by identical links. Let us assume that intra-cluster communication speed is $S$ while the inter cluster speed is $s$. Let us define a parameter $r = \frac{S}{s}$. Let us consider a case where a certain parallel job is processed in one cluster and the duration of the communication phase is $t$. If the job is instead co-allocated, then we have two types of communication.

(i) The intra-cluster communications where the communicating processes are in the same component. Such a communication takes place within the same cluster. The duration of such a communication is $t$.

(ii) The inter-cluster communication where the communicating processes are in different components. Such a communication takes place between clusters. The duration of such a communication is $rt$.

Since in synchronous communication a processing phase has to wait for the completion of all communications in the preceding communication phase, then the effective duration of the job communication phase is $max(t, rt)$. In practice $s < S$ implying that $r > 1$. Therefore, the duration of the communication step increases to $rt$ when the job is co-allocated. Co-allocation therefore comes with a negative effect on performance. Co-allocated jobs
occupy the processors longer than they would on a single cluster. On top of the negative effect, co-allocation breaks up the jobs to components. The smaller components come with the packing benefits studied in Chapter 3. The net effect of co-allocation is therefore the resultant of the packing benefit and the communication penalty. If the penalty is greater than the benefit, co-allocation is unviable otherwise it is viable.

5.2 Related Work

Ignoring communication is one of the common pitfalls in evaluation of parallel job scheduling [37]. It leads to artificially good but misleading deductions. In multi-cluster systems, it is more pronounced due to the relatively slow inter-cluster links. The slow inter-cluster links highly influence the execution time of the co-allocated job.

Bucur and Epema [7] and Bucur [10] considered an all-to-all synchronous communication model and used the fixed time penalty approach to model the effect of communication. They assumed that the slower inter-cluster links increase the execution time by a fixed percentage. They therefore increased the run time of all co-allocated jobs by a certain (fixed) percentage. This percentage is a representation of the intra-cluster to inter-cluster speed ratio ($r$). The penalty is independent of the number of components a job is broken into.

Sonmez et al. [75] considered an asynchronous communication case. The co-allocated job, like in [7], is given a run time penalty. However, the penalty suffered by the job is proportional to the amount of inter-cluster communication the job makes. The penalty suffered, therefore, depends on the number of components as well as the width of each component. In this approach, much as the value of $r$ is fixed, different jobs, depending on size, number of components and width of each component may have different execution time penalties.

Jones et al. [46][50] used bandwidth to model the effect of co-allocation on the job run time. They considered a case where inter-cluster process requires
a certain (fixed) amount of bandwidth to execute. Depending on the size of the job and the width of the components, the amount of bandwidth it requires to execute in the allotted time is computed. If the bandwidth available is less than the required bandwidth, then the execution rate of the job reduces. The extra execution time penalty is proportional to the bandwidth shortfall. Since the shortfall does not necessarily span the entire job run time, the total penalty is got by summing up the individual penalties depending on bandwidth fluctuations as the job executes.

5.3 Experimental Set Up

5.3.1 The multi-cluster system

We consider a system of 5 homogeneous clusters of 20 processors each. The clusters process by pure space slicing. The clusters are connected by dedicated wide-area links with some communication latency. The system is served by one queue and one scheduler and co-allocation is possible on it.

5.3.2 The job stream

We use a synthetic workload. We generate the workload from $D(0.85)$ over the interval $[1, 19]$. We consider exponentially distributed inter-arrival and execution times of means 10 and 0.54 respectively (this leads to a load of 0.786). All jobs to be co-allocated are broken into 4 components. For evaluation, we use four approximately equal size based groups $S_1$, $S_2$, $S_3$ and $S_4$. Their boundaries are the size lower quartile, median, and upper quartile. They have a numerical representation of 25.3%, 27.7%, 22.9% and 24.1% and load representation of 6.0%, 27.7%, 23.1% and 57.6% respectively.

5.3.3 Scheduling algorithm and placement policy

We use the FPFS algorithm with $\text{maxJumps} = 10$. For the placement policy, we use the Worst Fit policy.
5.4 Effect of Communication on Co-allocation

We now make a further study on the effect of communication on the performance of co-allocation. Like Bucur and Epema [7], we consider a synchronous communication model. Our approach differs from that of Bucur and Epema [7] by the way we model the penalty a co-allocated job suffers. Bucur and Epema [7] considered only the ratio of intra-cluster to inter-cluster link speed ($r$). We consider the communication intensity as well. Our view is that jobs collocated on the same multi-cluster system may experience different execution time penalties due to differences in communication intensity. The job communication intensity is dependant on the internal operations of the job. Broadly, communication intensity of a job determines the proportion of its execution time which is spent on communication. We study the bounds of co-allocation viability putting communication into consideration. We study the interpretational challenges on the viability bounds. We then study the effect of communication intensity heterogeneity with in the job stream on performance/viability of co-allocation.

5.4.1 The communication model

Our communication model is similar to that of Bucur and Epema [7]. We consider an all-to-all synchronous communication model among all tasks in a co-allocated job. The execution time penalty suffered by a co-allocated job has a (relative) link speed and communication intensity component. Let us consider a job $J_i$ with $n_{ci}$ communication steps and $n_{pi}$ processing steps. We assume that for a certain job, all communication (processing) steps have the same duration. However, the duration of a communication step is not necessarily equal to that of the processing step. Let us assume that if $J_i$ is processed on a single cluster, the duration of a single communication and processing step is $t_{ci}$ and $t_{pi}$ respectively. We can therefore define the total time $J_i$ takes during communication and processing to be $T_{ci}^i$ and $T_{pi}^i$ where

$$T_{ci}^i = n_{ci}^i \times t_{ci}^i$$

(5.1)
and

\[ T_i^p = n_i^p \times t_i^p \] (5.2)

The total execution time \( T_i^e \) of \( J_i \), on a single cluster, is got by adding the processing and communication components. Therefore

\[ T_i^e = n_i^c \times t_i^c + n_i^p \times t_i^p = T_i^c + T_i^p \] (5.3)

This is the same expression of \( T_i^e \) used in some of the previous related research [45],[51]. Let us assume that the communication component \( T_i^c \) constitutes a proportion \( \alpha_i \) of the total execution time of \( J_i \). The parameter \( \alpha_i \) \((0 < \alpha_i < 1)\) represents the communication intensity of \( J_i \). If \( \alpha_i \) tends to 1, then \( J_i \) is communication intensive and if it tends to 0, then it is processor intensive. We can therefore rewrite Equation (5.3) as

\[ T_i^e = \alpha_i T_i^e + (1 - \alpha_i) T_i^e \] (5.4)

If \( J_i \) is co-allocated, some of the communication has to take place across the (slower) inter-cluster link. The value of \( t_i^c \) is not the same for all the communications in \( J_i \). Effectively, co-allocation increases the communication component of \( T_i^e \). Though some of the communication (within components) remains intra-cluster, the inter-cluster communication determines the actual duration of the communication steps. This is because every processing step has to wait until all processes complete the communication step. This further implies that for a co-allocated job, the penalty suffered is independent of the proportion of the intra-cluster messages (and hence number of components). It only depends on the duration of the slowest message.

Let us define a parameter \( \lambda \) \((\lambda > 0)\) which is in such a way that the inter-cluster turn around time of a message exceeds the intra-cluster turn around by a factor \((1 + \lambda)\). This implies that when a job is co-allocated, the \( \alpha_i T_i^e \) component is effectively increased by a factor \((1 + \lambda)\). If the run time of \( J_i \) when co-allocated is \( \tau_i^e \), then

\[ \tau_i^e = (1 + \lambda)\alpha_i T_i^e + (1 - \alpha_i) T_i^e = (1 + \alpha_i \lambda) T_i^e \] (5.5)
5.4. **EFFECT OF COMMUNICATION ON CO-ALLOCATION**

We can therefore define $\psi_i = \alpha_i \lambda$ and rewrite Equation (5.5) as

$$\tau_{ie} = (1 + \psi_i)T_{ie}$$

(5.6)

Since $0 < \alpha_i < 1$ and $\lambda > 0$, then $\psi_i > 0$. The penalty in Equation (5.6) is similar to the fixed time penalty approach employed in [7][45][51]. However, our penalty compounds the job ($\alpha_i$) and link ($\lambda$) characteristics. In [7], only job communication intensity was considered while in [45][51] only the per-processor bandwidth requirements of the job were considered. While we assume links to be uniform, the communication intensity can be different for different jobs. The parameter $\psi$ therefore follows the distribution of $\alpha$.

### 5.4.2 Viability of co-allocation

We now investigate the parameter values with in which co-allocation is viable. We consider a scenario where the value of $\psi$ is the same for all the jobs (the case of varying $\psi$ is studied in Section 5.4.4). We first consider a case where jobs are not broken into components and they are scheduled without co-allocation. We then consider a case where $\text{thres} = 11$. Every job whose size is greater than $\text{thres}$ is broken into components and co-allocated. In the later case, we study the performance variation with $\psi$. Our studies consider the trend for the entire job stream as well as the job groups. We summarize the trends in Figure 5.1. From Figure 5.1, we observe that an increase in $\psi$ leads to poorer performance for all the jobs. The performance of large jobs deteriorates at a higher rate than that of small jobs. When we compare the co-allocation with no co-allocation case, we observe that co-allocation is viable if the value of $\psi$ is low. If $\psi$ is high, it is of performance benefit not to co-allocate jobs. We also observe that the threshold value of $\psi$ beyond which co-allocation is not viable is not the same for all the jobs. While the entire average has a value of 0.815, $S_4$ has a 0.160 and it is over 0.3 for $S_1$. 
5.4.3 The effect of \textit{thres}

In Section 5.4.2, we considered a fixed value of \textit{thres} when investigating co-allocation viability. We now investigate how \textit{thres} (and \textit{ψ}) affect the performance of a co-allocated job stream.

\textbf{The effect of \textit{thres} at fixed \(ψ\)}

We set \(ψ = 0.05\) and vary the value of \textit{thres} and study the performance trends for the different groups. We summarize the results in Figure 5.2. We observe that increasing \textit{thres} leads to poorer performance of all the job groups. Groups with large jobs register a higher rate of performance deterioration with increasing \textit{thres} compared to groups with small jobs.

\textbf{The effect of \textit{thres} on different \(ψ\) values}

In this section, we use only \(S_4\) jobs. This is because \(S_4\) jobs have the highest rate of performance deterioration with \textit{thres} (Figure 5.2). \(S_4\) jobs also have the lowest value of \(ψ\) beyond which co-allocation is not viable. This implies that in case co-allocation is viable for \(S_4\) jobs, it is viable for the rest of the
5.4. EFFECT OF COMMUNICATION ON CO-ALLOCATION

Figure 5.2: Performance variation with \( thres \) for the different groups

job stream (and vice versa).

Figure 5.3 shows the performance variation of \( S_4 \) jobs with \( thres \) for different values of \( \psi \). We observe that for every \( \psi \) value, there is an optimal \( thres \) value \( thres^* \) where if all jobs with size greater than \( thres^* \) are co-allocated, we get the best performance. The value of \( thres^* \) is low for low values of \( \psi \) and high for high values of \( \psi \). This implies that when there is a small execution time penalty from communication, it is beneficial to break up many jobs. However, if the penalty is higher, it is not.

The effect of \( thres \) on different loads

Using \( \psi = 0.1 \), we use different mean inter-arrival times to vary the load of the job stream. We investigate the performance variation of \( S_4 \) jobs with \( thres \) for the different loads. We summarize our results in Figure 5.4. We observe in Figure 5.4 that increasing the load leads to poorer performance. We also observe that performance trend keeps the same.
5.4.4 The effect of communication dispersion

All our studies in this Chapter assumed a fixed $\psi$ for all co-allocated jobs. This implies that the intra-cluster to inter-cluster speed ratio (hence $\lambda$) is fixed. It also implies that all jobs have the same communication intensity ($\alpha$). In practice, it is easy to ensure that $\lambda$ is fixed. This is because it only calls for homogeneity among the intra-cluster and inter-cluster networks. However, it is hard to ensure that $\psi$ is fixed for the entire job stream. We are not aware of any documented studies on the distribution of communication intensities in typical supercomputer/multi-cluster workloads. Nevertheless, it is our belief that job communication intensities are not fixed. This is due to the diversity of the sources and applications processed by multi-cluster systems. We therefore assume $\lambda$ to be fixed but $\alpha$ to vary with jobs. For our studies, we consider a situation where $\psi \sim U[0.001, 0.199]$ (mean = 0.1). We study performance variation with *thres* and compare it with a case of fixed $\psi = 0.1$. We also study the relative performance of communication based jobs groups.
5.4. EFFECT OF COMMUNICATION ON CO-ALLOCATION

Figure 5.4: Performance variation with \(\text{thres}\) for different load values

The effect of dispersion on performance

Figure 5.5 shows the performance variation of the different job groups with \(\text{thres}\). It consists of the same groups in cases when \(\psi = 0.1\) and \(\psi \sim U[0.001, 0.199]\). We observe that if the \(\psi\) value follows a uniform distribution, all job groups perform poorer than when \(\psi\) is fixed. This implies that increasing dispersion in the communication characteristics of the job stream leads to poorer performance. We further observe that the negative effect of the communication heterogeneity goes beyond the co-allocated jobs. Jobs in \(S_1\) and \(S_2\) are not co-allocated but are negatively affected by the dispersion in \(\psi\).

Relative performance of communication based groups

We also make communication based group-wise performance studies. We consider a case where \(\psi \sim U[0.001, 0.199]\) and generate job groups using the value of \(\psi\) for the job. These groups are only made from the co-allocated jobs. We make four communication based groups \(C_1, C_2, C_3\) and \(C_4\). They
are made up of jobs where $\psi$ is in the ranges of $(0, 0.5)$, $(0.5, 1.0)$, $(1.0, 1.5)$ and $(1.5, 2.0)$ respectively. We investigate the group-wise performance variation $\text{thres}$ and summarize our results in Figure 5.6. We observe that there are little differences among the performance of communication based groups. We also observe that they follow the same trend with changing $\text{thres}$. This implies that the effect of communication is largely felt by all the jobs in the job stream rather than the individual jobs in proportion to the run time penalty suffered.

### 5.5 Discussion and Implications

#### 5.5.1 Penalty representation

We have modeled the execution time penalty using two parameters; the job stream parameter ($\alpha$) and the link parameter $\lambda$. The job stream parameter represents communication intensity and the link parameter represents the intra-cluster speed relative to the inter-cluster speed. The two parameters compound to a single parameter $\psi$ which represents the execution time penalty suffered by a co-allocated job. In terms of representation, our ex-
5.5. DISCUSSION AND IMPLICATIONS

Figure 5.6: Performance variation with *thres* for communication based groups

eecution time penalty is similar to that used in previous research [7]. Our penalty representation, however, extends the interpretation of co-allocation viability beyond its interpretation in [7]. In our case, co-allocation can be viable on any inter-cluster link so long as the communication intensity is low enough. Likewise, co-allocation can be viable on any job stream so long as the inter-cluster speed is fast enough. This extends the boundary definition in [7] which is only dependent on the relative link speeds.

5.5.2 Group-wise performance trends

We have observed that small jobs, despite not suffering an execution time penalty deteriorate in performance when $\psi$ is increased. This can be attributed to the implication of the execution time penalty on the availability of processors to jobs in the queue. If a job is co-allocated, its execution time increases. This implies that it spends more time on the processors. This holds both small and large jobs in the queue since the processors are not yet freed. Increasing $\psi$ implies that the co-allocated jobs will run for a longer
CHAPTER 5. COMMUNICATION AND CO-ALLOCATION

time and hence the jobs will be held for a longer time in the queue. The effect of co-allocation therefore affects the job being processed (by executing for longer) and those in the queue (by delaying the start time). Due to the second factor, jobs that are not co-allocated also experience deterioration in performance.

5.5.3 Group-wise relative performance

We have observed that there are differences in group-wise performance with small jobs performing better than large jobs. The performance gap increases as $\psi$ increases. This can be attributed to:

(i) schedulability;

(ii) the extra time spent executing; and

(iii) the extra time spent in the queue due to extra execution time of a running job.

Other factors fixed, factor (i) explains the relatively better performance of small jobs compared to large jobs since small jobs are more schedulable. Factor (ii) affects only co-allocated jobs and factor (iii) affects all jobs equally. Factors (ii) and (iii) increase with $\psi$. The group-wise performance differences, therefore, increase with $\psi$.

5.5.4 Group-wise viability of co-allocation

We have observed that different job groups have different thresholds of co-allocation viability. The group-based viability thresholds show us that the entire job stream threshold may not be a realistic choice to determine co-allocation viability. For example, for $\psi$ values between 0.160 and 0.815 (Figure 5.1), co-allocation is viable when seen from the entire job stream point of view but not viable when seen from $S_4$ point of view. Much as $S_4$ jobs constitute 24% of the jobs, they constitute 57.6% of the load. This implies that taking co-allocation to be viable when $\psi > 0.16$, we do it when actually over
half of the load is at a performance disadvantage. Analyzing co-allocation in job groups helps make more realistic decisions. This is in line with previous studies in group-wise performance analysis [35][76][77][78].

### 5.5.5 The role of \( \text{thres} \)

We have observed that the value of \( \text{thres} \) has a big impact on the performance of co-allocation. Performance wise, the value of \( \text{thres} \) has positive and negative implications. On the positive side, it determines how many jobs will be broken into components. This has a high effect on the schedulability of the entire jobs stream [59]. On the negative side, it determines what proportion of the jobs will suffer the execution time penalty (the penalty depends on \( \psi \)). Extending the execution time of a job implies it will occupy the processors for a longer time and hence extending the delay of the jobs in the queue. The net performance is therefore dependant on how the two relate. If there is minimal penalty (low \( \psi \)), maximizing schedulability (by lowering \( \text{thres} \)) leads to good performance (Figure 5.4). If \( \psi \) is high, \( \text{thres} \) needs to be high enough to reduce the extra processor hours required for execution time penalty but low enough to get the packing benefits of breaking up jobs. We therefore get an optimal \( \text{thres} \) value for each \( \psi \) value. This optimal value is independent of the load (Figure 5.5).

### 5.5.6 The effect of communication heterogeneity

We have observed that communication heterogeneity within the job stream leads to poor performance for all the job groups. This can be attributed to the fact that different extents of execution time penalty leads to more fragmentation which leads to poorer performance. The fragmentation keeps jobs for more time in the queue which implies that the negative effect is felt by the entire job stream. The schedulability of the jobs however is independent of its execution time (which is unknown in our case). This leads to a situation where the communication based job groups have little performance differences among them.
5.5.7 Overall viability of co-allocation

Previous studies on co-allocation viability considered co-allocation when the viability thresholds are based on the entire job stream average. Our consideration of job groups show that the entire job stream threshold could actually be unrealistically high. Previous studies also considered fixed penalty for all co-allocated jobs. Our study on the effect of communication heterogeneity show a poorer performance of co-allocation compared to the fixed penalty case. This implies that in reality, owing to skewed load distribution in the job streams and communication heterogeneity among jobs, co-allocation may not be as viable as previously.

5.6 Conclusion

In this chapter, we have studied the effect of communication on the performance of co-allocation. We consider a synchronously communicating job stream. We use the effect of the slower wide area network and job communication intensity to model the execution time penalty. This extends the use of only the intra-cluster to inter-cluster speed ratio. It also extends the practical implications of the viability of co-allocation as with respect to relative intra-cluster and inter-cluster speeds. We have studied the viability of co-allocation as viewed from the job group and heterogeneity point of view. We extend the interpretation of co-allocation viability beyond the job stream averages to job group averages. We have also studied sensitivity of co-allocation with selected job stream parameters. This included the threshold job size to allow co-allocation and the communication intensity of the co-allocated jobs. We have also studied the effect of load on the viability of co-allocation as well as the performance trends. Finally, we have studied the effect of communication heterogeneity on performance of co-allocation as well as relative performance for communication based groups. We have also discussed the implications of our results on the practice of co-allocation.
Outline: In Chapter 3 and Chapter 4, we observed that sometimes, there are big performances difference among jobs. This can be due to selective starvation/discrimination of jobs by schedulers. In this chapter, we study the concept of fairness in parallel job scheduling. We introduce the concept and relevance of fairness in Section 6.1. In Section 6.2, we review approaches/metrics used to evaluate fairness in parallel job schedulers. In Section 6.3, we examine how the metrics represent job discrimination/favoritism. We identify cases where discrimination/favoritism is not adequately represented by the metrics. In Section 6.4, we discuss scenarios where existing metrics may misrepresent (un)fairness and propose checks to handle them. Finally, we conclude the chapter in Section 6.5. This chapter is based on work in [63][66].

6.1 Introduction

6.1.1 The importance of fairness

Fairness is an important factor in all queuing systems. Possibly, it is because of fairness that queues came up in the first place. Parallel job scheduling has been extensively studied [26]. However, most of the studies were focused on performance rather than on fairness. Unrealistic deductions from performance metrics mostly stem from scheduler unfairness [37]. This is because some jobs may be favored while others are discriminated. In typical super-
computer workloads [19][56], small jobs make up the majority of the jobs and the minority of the load. Favoring small jobs gives an impressive job stream performance despite poor performance of the large jobs. Fairness, ideally, is used to measure the extent of favoritism and discrimination by the scheduler. A good metric of fairness evaluation would identify schedulers whose apparent good performance is due to starvation of some jobs at the expense of others. Studies by Refaeli et al. [88][89] showed that people in queues are more annoyed by perceived unfairness than by the actual (poor) performance. An unfair system with apparently good performance does not offer satisfaction to the users. Fairness therefore has a large contribution to user satisfaction.

6.1.2 The concept of fairness

Fairness has roots from social justice. It is based on the idea of equity. Resources/load need to be distributed appropriately for the scheduler to be fair. Due to differences in resource requirements and seniority, appropriate resource distribution is not necessarily the same as equal resource distribution.

In parallel job scheduling, fairness can be looked at from the system or user perspective. From the system perspective, a scheduler is fair if it does not favor some servers (like clusters) over others when allocating loads. Likewise, some clusters need not to be (comparatively) overloaded by virtue of being part of the multi-cluster system. Jones et al. [52] used the concept of varying cluster loads to illustrate (un)fairness from a system point of view. If one of the clusters in a multi-cluster system has fewer incoming jobs than others, its load will have to be 'topped up' by jobs from other clusters. It therefore caters for more load than what it would if it was a stand alone cluster. Its users (owners) may also experience poorer service than a case if it was a standalone cluster. In such a case, it appears that joining the multi-cluster system came with more load to the cluster and poorer performance to the cluster owners. In such a case, the scheduler is unfair from a system point
of view. Sabin et al. [70] also looked at fairness from the system point of view. They proposed mechanisms like increased local priority so that owners of less loaded clusters do not get big deteriorations in performance.

From the user’s point of view, fairness is looked at in terms of discrimination and favoritism among different users/jobs. The scheduler should not give a higher scheduling opportunity to one job at the expense of others. Likewise, it should proportionately distribute the available resources to the competing jobs. When measuring fairness from a user’s point of view, we look for evidence of job starvation/favoritism or un-proportionate distribution of system resources among the jobs. This should be in line with what the job would actually be entitled to in an ideal situation.

Even in ideally fair cases, jobs are not expected to have equal performance. The differences in performance can be dictated by factors like schedulability and the overall traffic at the moment of job arrival. Intentionally delaying the processing of a job (e.g. a small job) so that its performance is the same as that of a large job is obviously unfair. At the same time, it is unfair to starve jobs further (e.g large jobs) at the expense of other more schedulable ones.

In our study, we limit ourselves to fairness from a user’s point of view. We therefore look at performance, favoritism and discrimination as seen by the user in a parallel job scheduling set up.

6.2 Fairness in Parallel Job Scheduling

Though fairness is applicable in all queuing systems, it is envisaged differently in specific queuing environments. This is due to differences in the characteristics of the queuing set up considered. In this section, we briefly discuss the perception and measurement of fairness in parallel job scheduling as reported from previous studies.
6.2.1 Perceptions of fairness

There are two ways in which fairness can be viewed from the user’s perspective. One considers equal distribution of resources among the participating jobs. This approach can be provided by processor sharing where each job, in a round-robin way, is given an equal fraction of processing power in a single processor computer. Chiang et al. [17] implemented this approach on parallel computers. The second approach uses the user’s expected order of service to evaluate favoritism and discrimination and hence fairness. It has been considered in studies like [39][71][96]. The first approach is implementable in time slicing systems. Since we consider dedicated processing in our work, we limit ourselves to the second approach.

6.2.2 Approaches to fairness evaluation

The most popular approaches to fairness in parallel job scheduling fairness evaluation include dispersion [39][96], fair start time analysis [71][72] and the resource allocation queuing measure (RAQFM)[4][69].

The dispersion approach

In the dispersion approach, the statistical measures of dispersion among the performance of jobs are used. If we consider a metric used to measure performance in a certain system, we evaluate fairness by using the performance dispersion among the jobs’ performance. The measures used include standard deviation $\sigma$, variance $\sigma^2$ and coefficient of variation ($C_V$). If the extent of dispersion is low, it implies that there is less favoritism and discrimination among the jobs by the scheduler. This implies that the scheduler is fair. If there is more dispersion, then it implies that some jobs are favored while others are discriminated. This implies that the scheduler is unfair.

Jain et al. [39] proposed that a good measure of fairness should be continuous, bounded, scale independent and population size independent. They therefore propose the fairness coefficient $\kappa = \frac{1}{1 + C_V}$ which satisfies these
characteristics. If a scheduler has \( \kappa = 1 \), then it is ideally fair and in case \( \kappa = 0 \), then it is unfair.

**Fair-start time analysis**

The fair-start time approach was proposed in [71]. The idea behind the approach is that if a later arriving job delays the processing of another job that arrived before it, the delayed job is unfairly treated. Fairness is therefore measured by the extent to which jobs are delayed by others that arrived after them. To get the extent of unfair treatment, we get the difference between the *fair-start* time and the *actual start* time of the job. To get the fair start time \( t_f^i \) of job \( J_i \), the job stream is truncated at \( J_i \) (implying no job arrived after \( J_i \)) and scheduled. The time \( J_i \) starts processing is its fair-start time. To get the actual start time \( t_a^i \) of \( J_i \), the entire job stream is scheduled. The time \( J_i \) starts processing is its actual start time. If \( t_f^i < t_a^i \), then \( J_i \) is unfairly treated by the scheduler. The average of \( t_a^i - t_f^i \) for all unfairly treated jobs is used to evaluate scheduler fairness.

**RAQFM**

In RAQFM [4][69], the underlying principle is that all jobs in the queue are entitled to an equal share of the system resources. If at a time \( t \) there are \( N(t) \) jobs in the queue, then each job is entitled to \( \frac{1}{N(t)} \) of the system resources. If a job \( J_i \) is given a proportion \( s_i(t) \) of the resources, then the temporal discrimination \( d_i(t) \) is given by \( d_i(t) = s_i(t) - \frac{1}{N(t)} \). If \( J_i \) arrives in the queue at time \( t_i^A \) and finishes processing at time \( t_i^F \), then the total discrimination \( D_i \) is given by

\[
D_i = \int_{t_i^A}^{t_i^F} S_i(t) - \frac{1}{N(t)} \, dt
\]  

The overall discrimination of the system is got by computing the variance of the total discriminations of the jobs \( \text{Var}[D] \). This is because for non-idling systems, the mean discrimination \( E[D] \) is always equal to 0.
6.3 Metrics’ Representation of Discrimination

Though fairness metrics are meant to represent the existence (or absence) of discrimination/favoritism, there are cases where the implied discrimination is not discrimination in practice (and vice versa). This can lead to scenarios where the implied unfairness is not unfairness in practice. In such a scenario, the fairness metric can be considered inappropriate. In this section, we examine the three approaches to fairness evaluation and identify cases where the implied unfairness is not in line with what would be felt as unfairness in practice.

6.3.1 Dispersion

In this approach, we compute the performance dispersion with in the job stream to measure fairness. Low dispersion implies a fair scheduler and high dispersion implies an unfair scheduler. Deductions from measures of dispersion may be misleading since sometimes the increase in dispersion is not due to starvation.

To illustrate a case where more dispersion is not necessarily an implication of more starvation, let us consider online jobs \( J_1, J_2, \ldots, J_N \). In the first instance they are scheduled by FCFS and in the second instance they are scheduled by Conservative Backfilling (CB). Let us consider the performance metric to be Average Waiting Time (AWT). When scheduled by FCFS, \( AWT = \mu, \text{ variance} = \sigma^2 \) and a job \( J_i \) has waiting time \( t_w^i \). When scheduled by CB, \( AWT = \mu', \text{ variance} = \sigma'^2 \) and the waiting time for \( J_i \) is \( t_w'^i \). We compare the individual job performance, the average job stream performance and the extent of dispersion among the jobs’ performance (fairness).

**Theorem 6.3.1** \( t_w^i \geq t_w'^i \ \forall \ J_i \ i = 1, 2 \ldots N \)

**Proof** If no job backfills, CB \( \Rightarrow \) FCFS. Therefore:

\[ t_w^i = t_w'^i \ \forall J_i \ i = 1, 2 \ldots N \]

In CB, there are three possible ways a job can get performance benefits
6.3. METRICS’ REPRESENTATION OF DISCRIMINATION

(i) **Direct benefit**: In this case, a job jumps and gets processed at an earlier time;

(ii) **Indirect benefit**: When a job jumps in (i), the queue is shortened. The jobs initially behind it get closer to the head of the queue and may get improved reservation. Jobs yet to arrive also find a shorter queue. They get an improved reservation in advance; and

(iii) **Combined benefit**: A job gets the benefit in (ii) due to backfilling of the jobs ahead of it and also backfills to get the benefits of (i).

If set $S$ consists of jobs directly or indirectly affected by backfilling, then $t^w_i > t^{w'}_i \forall J_i \in S$

Since a job either benefits from CB or retains its FCFS reservation time:

$t^w_i \geq t^{w'}_i$  

**Theorem 6.3.2** $\mu \geq \mu'$

**Proof** By definition:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} t^w_i \text{ and } \mu' = \frac{1}{N} \sum_{i=1}^{N} t^{w'}_i$$

From the validity of Theorem 6.3.1, Theorem 6.3.2 is also valid.

Let us define two non-negative parameters $\delta \mu$ and $\delta t_i$ as:

$$\delta \mu = \mu - \mu' \text{ and } \delta t_i = t^w_i - t^{w'}_i \quad (6.2)$$

**Lemma 6.3.1** If we break the job stream into two disjoint sets $S_1$ and $S_2$, where $J_m \in S_1$ iff $t^w_m \leq \mu$ and $J_n \in S_2$ iff $t^w_n > \mu$, then:

$$\left| \sum_{J_m \in S_1} (\mu - t^w_m) \delta t_m \right| \geq \left| \sum_{J_n \in S_2} (\mu - t^w_n) \delta t_n \right|$$
Proof Since \( \mu \) is a measure of central tendency for \( t_1^w, t_2^w, \ldots, t_N^w \), then:

\[
\sum_{J_m \in S_1} (\mu - t_m^w) = \sum_{J_n \in S_2} (\mu - t_n^w) \tag{6.3}
\]

In FCFS, a job keeps in queue due to two reasons:

(i) it has not reached the head of the queue; or

(ii) it is at the head of the queue but cannot fit in the available free processors.

Reason (i) affects all jobs equally while reason (ii) affects large jobs more. This is because small jobs easily accumulate the required processors to start execution. In fact, they can easily process of initially idle processors (caused by fragmentation). Small jobs therefore perform better than large jobs. This implies that small jobs dominate \( S_1 \) and large jobs dominate \( S_2 \).

CB utilizes processors that would be idle and small jobs mostly benefit from it [78]. Small jobs therefore get bigger performance improvement compared to large jobs. This implies that (i) for jobs in \( S_1 \), \( |\mu - t_m^w| > 0 \) and \( \delta t_m \gg 0 \) and (ii) for jobs in \( S_2 \), \( |\mu - t_n^w| > 0 \) and \( \delta t_n \approx 0 \).

Multiplying \( \mu - t_m^w \) with a relatively large \( \delta t_m \) makes LHS grow (absolutely) faster than RHS where \( \mu - t_n^w \) is multiplied with a smaller \( \delta t_n \).

**Theorem 6.3.3** \( \sigma^2' \geq \sigma^2 \)

**Proof** From the definition of variance:

\[
\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (t_i^w - \mu)^2 \quad \text{and} \quad \sigma^2' = \frac{1}{N} \sum_{i=1}^{N} (t_i'^w - \mu')^2 \tag{6.4}
\]

From (6.4), \( \sigma^2' - \sigma^2 \) can be written as:

\[
\frac{1}{N} \sum_{i=1}^{N} \left[ (t_i'^w - \mu')^2 - (t_i^w - \mu)^2 \right] = \frac{1}{N} \sum_{i=1}^{N} (t_i'^w - \mu - t_i^w + \mu)(t_i'^w - \mu' - t_i^w + \mu)
\]
6.3. METRICS’ REPRESENTATION OF DISCRIMINATION

eliminating $t^w_i$ and $\mu'$ using (6.2), we get:

$$
\frac{1}{N} \sum_{i=1}^{N} (t^w_i - \delta t_i - \mu + \delta \mu + t^w_i - \mu)(t^w_i - \delta t_i - \mu + \delta \mu - t^w_i + \mu)
$$

$$
= \frac{1}{N} \sum_{i=1}^{N} (2t^w_i - 2\mu + \delta \mu - \delta t_i)(\delta \mu - \delta t_i)
$$

$$
= \frac{1}{N} \sum_{i=1}^{N} (2(t^w_i - \mu) + \delta \mu - \delta t_i)(\delta \mu - \delta t_i)
$$

$$
= \frac{1}{N} \sum_{i=1}^{N} (2(t^w_i - \mu)(\delta \mu - \delta t_i) + (\delta \mu - \delta t_i)^2)
$$

$$
= \frac{1}{N} \sum_{i=1}^{N} (\delta \mu - \delta t_i)^2 + \frac{2}{N} \sum_{i=1}^{N} (t^w_i - \mu)(\delta \mu - \delta t_i)
$$

$$
= \frac{1}{N} \sum_{i=1}^{N} (\delta \mu - \delta t_i)^2 + \frac{2}{N} \sum_{i=1}^{N} (t^w_i \delta \mu - t^w_i \delta t_i - \mu \delta \mu + \mu \delta t_i)
$$

$$
= \frac{1}{N} \sum_{i=1}^{N} (\delta \mu - \delta t_i)^2 + \frac{2}{N} \left[ \sum_{i=1}^{N} t^w_i \delta \mu - \sum_{i=1}^{N} t^w_i \delta t_i - \sum_{i=1}^{N} \mu \delta \mu + \sum_{i=1}^{N} \mu \delta t_i \right]
$$

$$
= \frac{1}{N} \sum_{i=1}^{N} (\delta \mu - \delta t_i)^2 + \frac{2}{N} \left[ N \mu \delta \mu - \sum_{i=1}^{N} t^w_i \delta t_i - N \mu \delta \mu + \sum_{i=1}^{N} \mu \delta t_i \right]
$$

$$
= \frac{1}{N} \sum_{i=1}^{N} (\delta \mu - \delta t_i)^2 + \frac{2}{N} \left[ \sum_{i=1}^{N} \mu \delta t_i - \sum_{i=1}^{N} t^w_i \delta t_i \right]
$$

This simplifies to:

$$
\sigma'^2 - \sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (\delta \mu - \delta t_i)^2 + \frac{2}{N} \sum_{i=1}^{N} (\mu - t^w_i)\delta t_i
$$

(6.5)

Using the two sets defined in Lemma 6.3.1, Equation (6.5) can be rewritten as:

$$
\sigma'^2 - \sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (\delta \mu - \delta t_i)^2 + \frac{2}{N} \left[ \sum_{m \in S_1} (\mu - t^w_m)\delta t_m + \sum_{n \in S_2} (\mu - t^w_n)\delta t_n \right]
$$

(6.6)

We observe that $(\delta \mu - \delta t_i)^2 > 0$ and using Lemma 6.3.1, the RHS of Equation (6.6) is non negative.  

Theorem 6.3.4 If \( \kappa \) and \( \kappa' \) are the fairness coefficients for FCFS and CB respectively, then \( \kappa' \leq \kappa \)

Proof By definition, \( \kappa = (1 + C_V^2)^{-1} \), from Theorem 6.3.3 and 6.3.2:

\[
\sigma'^2 \geq \sigma^2 \Rightarrow \sigma' \geq \sigma
\]
and \( \mu \geq \mu' \)

Multiplying the two inequalities imply

\[
\frac{\mu \sigma'}{\mu'} \geq \frac{\sigma}{\mu}
\]

Therefore

\[
C_V' \geq C_V \Rightarrow C_V'^2 \geq C_V^2
\]
Since \( \kappa = \frac{1}{(1+C_V^2)} \)

\[
\kappa' \leq \kappa \quad \blacksquare
\]

Theorem 6.3.1, Theorem 6.3.3 and Theorem 6.3.4 show that:

(i) No job in CB perform worse than in FCFS, therefore CB does not starve any job compared to FCFS;

(ii) Dispersion in CB is higher than that in FCFS.

A higher dispersion, therefore, does not necessarily imply that the scheduler favors some jobs at the expense of others. In parallel job scheduling, some of the factors that can lead to increased performance dispersion other than starvation include:

(i) Schedulability:

Jobs have different levels of schedulability [59]. This is due to the differences in resource required from the system. Highly schedulable jobs perform better than jobs which are less schedulable. This leads to differences in performance hence dispersion.
6.3. METRICS’ REPRESENTATION OF DISCRIMINATION

(ii) *The nature of the performance metric:*

Dispersion wise, different metrics that have the same performance implication can have different fairness implications. If for example we schedule a job stream at a very low load, every job gets scheduled immediately it arrives. This implies ideal fairness (no discrimination/favoritism). If we measure performance by waiting time, we get dispersion of 0 (ideal fairness). However, if we use response time to measure performance, we get non-zero dispersion (equal to dispersion of the job execution times). Performance wise, waiting time and response time can be used interchangeably in dedicated schedulers because they have the same performance implications. Nevertheless, they have different implications of fairness.

(iii) *Job arrival patterns:*

Studies in workload characteristics show variations in arrival patterns. These are in terms of daily and weekly peaks. A job arriving at peak hour will have poorer performance compared to that arriving at an off-peak hour. The poorer performance in this case is not necessarily due to starvation. This leads to performance dispersion which is not a result of starvation.

Performance dispersion and fairness do not necessarily imply each other. Using dispersion to deduce starvation may, in some cases, lead to wrong deductions.

6.3.2 Fair start time approach

Fair start time analysis looks deeper into the unfair treatment at individual job level. This approach, however, takes into account the delay suffered by the job relative to its neighbors. It does not explore the global effect of scheduler decisions on the delays a job suffers. In so doing, it does not fully consider the benefits/setbacks jobs get throughout their stay in the queue. The scheduler, therefore, may be partially evaluated by the metric. Tsafrir and Feitelson [91] observed that a decision made by a scheduler on a job can
have far reaching effects. As a job is backfilled for example, those ahead of it may get a delay and those behind it may get a benefit in reservation. The job delay, towards the end of the queue, is not necessarily the net delay caused by scheduler decisions.

![Figure 6.1: System status before scheduling jobs $J_k, J_{k+1 \ldots}$](image)

To illustrate this, let us consider a six node cluster scheduled up to the $(k-1)^{th}$ job represented in Figure 6.1. Let us consider a queue where the $k^{th}$, $(k+1)^{th}$ and $(k+2)^{th}$ jobs have sizes 4, 3 and 2 and run times 5, 2 and 4 respectively. Let us examine the local and global effect of the scheduler choice for the next job to schedule. Let us narrow the choice of the next candidate job to be either the $k^{th}$ or the $(k+1)^{th}$. The system state in each choice of the next job is shown in Figure 6.2

(i) **Case 1:** If $J_k$ is scheduled next, $J_{k+1}$ cannot start processing within the shown time frame.

(ii) **Case 2:** If $J_{k+1}$ is scheduled next, both $J_k$ and $J_{k+2}$ can start processing within the shown time frame.

Let us now focus on the reservations of jobs after $J_{k+2}$. Since in Case 2 $J_{k+1}$ is already processed by the end of the shown time frame, the queue is shorter by 1 job. This implies a performance benefit to all the jobs in the queue (and possibly those to come) compared to Case 1. This implies that a job can be delayed by others that arrived before it. Such gains/losses in performance are not catered for by the approach in [71].

This situation also helps us examine whether the delay in performance of $J_k$ in Case 2 should always be considered unfair treatment. It is possible that
6.3. METRICS’ REPRESENTATION OF DISCRIMINATION

![Figure 6.2: System states for Case 1 (top) and Case 2 (bottom)](image)

decisions on jobs that arrived before \( J_k \) improved/worsened its reservation time (the way choices of \( J_k \) and \( J_{k+1} \) affect jobs behind \( J_{k+2} \)). If it got a cumulative benefit of more than 1 unit of time, then it has a net benefit and therefore not unfairly treated by the scheduler. Since all the decisions are made by the scheduler, all their effects need to be put into consideration when evaluating the scheduler for fairness.

In fair-time analysis, we consider a job to be unfairly treated if it is delayed by another that arrived after it. Overall, the fair start time approach has the following short comings:

(i) Delays caused by earlier arriving jobs are not catered for.

(ii) Schedulers with different levels of satisfaction can be implied to be equally as fair. For example, when we use waiting time as a performance metric, FCFS and conservative backfilling are equally as fair. This is because in both cases, a job is never delayed by another that arrived after it. However, conservative backfilling outperforms FCFS.

(iii) The fair start time of a job is dependent on the scheduling algorithm and queuing policy. A single job therefore has different fair start times.
for different schedulers. The deviations, computed from these fair start times have different implications in real life.

(iv) The approach considers the disadvantage caused to a job by a later arriving job. This, in some cases, may be small and local that it does not give the global extent and shift in user satisfaction.

To compute the total delay/gain in performance a job gets, we need to add up all possible effects of scheduler decision on the job. Let us consider a job $J_k$ in a queue and examine circumstances that may alter its reservation time.

(i) Job $J_k$ may jump, get earlier processing. It gets a jump benefit $b_i^k$.

(ii) Jobs ahead of $J_k$ jump and get processed (in so doing utilize the would be idle processors). This shortens the queue and may improve the reservation of $J_k$. If $J_k$ gets a benefit $b_{ki}$ from the jumping of $J_i$, and all jobs that create such benefits to $J_k$ are in set $S^k_B$, then the extra benefit $\Delta R^1_k$ $J_k$ gets is given by

$$\Delta R^1_k = \sum_{\forall J_i \in S^k_B} b_{ki} \quad (6.7)$$

(iii) Like in (ii) above, jobs jump but are packed badly and increase fragmentation. This leads to a worse reservation of $J_k$. Let $J_i$ cause a disadvantage $d_{ki}$ to $J_k$. Let jobs that cause such a disadvantage be in set $S^k_D$. The total disadvantage $\Delta R^2_k$ is

$$\Delta R^2_k = \sum_{\forall J_i \in S^k_D} d_{ki} \quad (6.8)$$

(iv) Some of the jobs that arrived after $J_k$ jump and delay $J_k$ reservation. If such jobs are in a set $S^k_{D'}$ and $J_i$ causes a disadvantage $d'_{ki}$ to $J_k$, then the total disadvantage $\Delta R^3_k$ is

$$\Delta R^3_k = \sum_{\forall J_i \in S^k_{D'}} d'_{ki} \quad (6.9)$$
The net change in reservation for $J_k$ is got by computing the benefits in excess of the losses for $J_k$. If the net benefit is given by $\Delta R$, then

$$\Delta R = b_i^j + \sum_{\forall J_i \in S_B} b_{ki} - \sum_{\forall J_i \in S_D} d_{ki} - \sum_{\forall J_i \in S'_D} d'_{ki}$$

(6.10)

The approach in [71] used only $\Delta R^3_k$ to evaluate fairness. As shown in Equation (6.10), other factors are also involved. A disadvantage $\Delta R^3_k$ is not necessarily a net disadvantage. In some schedulers like aggressive backfilling, it rarely goes beyond a few jobs from the head of the queue [18]. Sometimes, however, the decision made on one job can have an effect that goes far deep into the queue [91]. Studies in [60] actually show that sometimes, even if the runtime is not considered, a big proportion of jobs achieve a net benefit in performance.

The contribution of $\Delta R^3_k$ can be substantial, negligible or non-existent depending on the scheduling environment. In cases where its substantial, fair-time approach gives a good estimate of starvation otherwise it does not.

6.3.3 RAQFM

RAQFM was proposed in [69] and analyzed in [4]. It uses the difference between the resources allocated to a job and that it is fairly entitled to deduce fairness. The RAQFM in [69] was meant for a single server facility. Extension to a multi-server facility is reported in [69] to be still ongoing. Its strengths include:

(i) It is able to intuitively explain the fair most decisions to be made in naturally challenging scenarios [4]. For example, it adequately handles a supermarket case where a customer with very few groceries is behind one with a lot of groceries;

(ii) It links the time spent in the queue with unfairness. When a job keeps unprocessed in the queue, $s_i(t)$ in Equation (6.1) is 0. This leads to more temporal discrimination which is realistic in practice; and
(iii) It caters for competition for resources since a higher number of jobs in the queue reduces the entitlement of each job (from the \( \frac{1}{N(t)} \) factor).

If used in parallel job scheduling, it does not cater for the time spent in the queue when making resource entitlements. It also does not put into consideration the resources required by the job. All jobs irrespective of the time spent in queue and the resources required have the same proportion of resource entitlement (\( \frac{1}{N(t)} \)). It can also claim discrimination where users get ideal service. We illustrate this by an example:

**Example 6.3.1** Let us consider two jobs needing 2 and 6 processors with runtime 5 each. They simultaneously arrive in a 10 node cluster where there is no other job in the queue and none is processing. This implies that they immediately get the service. RAQFM will consider the first job to have temporal discrimination \( \left( \frac{2}{10} - \frac{1}{2} \right) \times 5 = -\frac{3}{2} \) and the second job to have \( \left( \frac{6}{10} - \frac{1}{2} \right) \times 5 = \frac{1}{2} \). This implies the first job is discriminated while the second is favored yet both get ideal service.

This non ideal fairness implies than in some cases, RAQFM can also imply unfairness where it is not.

### 6.4 Discussion

Measures of fairness seek to evaluate discrimination/favoritism among the jobs by the scheduler. They therefore use evidence of discrimination/favoritism to measure scheduler fairness. However, we identify that there are non discriminative scenarios that can be viewed as discriminative. Likewise, the assumed ideal set up (like 0 dispersion) may not be the ideal in practice.

By the nature of typical parallel job streams, there are differences in job seniority, job size and schedulability as well as arrival traffic. Jobs are therefore not expected to have the same performance when scheduled in the ideal situation. The job discrimination/favoritism represented by measures of dispersion may have some background dispersion which is not discriminative in itself. This may be due to (i) the effect of differences in schedulability and
(ii) variations of the arrival traffic. For dispersion to effectively represent fairness, this background effect needs to be filtered out. However, since it is hard to numerically evaluate the background effect, it is also hard to filter it out.

Fair-start time approach considers the fact that the job has a time when it would fairly start processing. This time is not the same for the different jobs. It therefore caters for circumstances like traffic and job schedulability. However, the fair start time is dependent on the scheduler being evaluated. Since each job does not have a fixed reference point from which discrimination/favoritism is measured, it is hard to compare the levels of discrimination/favoritism. At the same time, the discrimination/favoritism considered is not the net disadvantage/advantage the job gets since some effects are not evaluated. So a job with a benefit can be considered discriminated and vice versa. There is therefore a need to evaluate if the unfair treatment recorded is actually the net not gross benefit/discrimination. This needs to be done against an ideally fair scheduler. Since the ideally fair scheduler is not known, it is also hard to evaluate the actual fair time.

For RAQFM, the proportion of the system resources a job is entitled to is the same for all jobs. It is independent of the seniority and the resource requirements of a job. This, in real life, is unfair itself. It can also consider an ideally scheduled set of jobs favored/discriminated.

6.5 Conclusion

In this chapter, we have made an evaluation of some of the approaches employed when evaluating fairness in parallel job scheduling. Specifically, we have evaluated the dispersion approach, the fair-start time approach and RAQFM. We have studied what is considered unfair by the metrics to what is actually unfair in practice. Our comparisons have been specific to the parallel job scheduling set up. We have observed that though some of the approaches (like dispersion and RAQFM) can be reliably used in other queuing paradigms, they have some loopholes in parallel job scheduling. Using
them can lead to misleading deductions. This was because what is represented as unfairness is not always unfairness. Like wise, unfairness my not be represented as unfairness.

There is therefore a need for improving the approaches to fairness evaluation so as to cater for the parallel job scheduling set up. This should include putting factors like job seniority, service requirements, and queue/system states. Implied discrimination (favoritism) need to be the actual discrimination (favoritism) as seen by the user. We propose such an approach to fairness evaluation in Chapter 7.
Outline: In Chapter 6, we observed that existing fairness metrics can imply unfairness where it is not. They can, therefore, lead to unrealistic conclusions when evaluating fairness in parallel job scheduling. In this chapter, we propose a new approach to fairness evaluation for parallel job schedulers. In Section 7.1, we describe the weaknesses in existing approaches addressed by our approach. In Section 7.2, we describe our approach and derive some metrics for the approach. This is followed by a description of the experiments to evaluate selected schedulers for fairness using our approach in Section 7.3. In Section 7.4, we use the most discriminated jobs and the worst performing jobs to validate our approach. We also discuss circumstantial appropriateness of the different metrics. We then conclude the chapter in Section 7.5. This chapter is based on work in [65].

7.1 Introduction

7.1.1 Background

In Chapter 6, we observed that what can be implied as favoritism/discrimination by some fairness metrics may not actually be favoritism/discrimination in practice. Therefore, the measures of fairness in parallel job scheduling may lead to misleading results. Nevertheless, fairness remains an important aspect of queuing systems in general and parallel job scheduling in particular.
There is a need to evaluate it in such a way that discrimination/favoritism is accurately and unambiguously represented. In such a situation, user satisfaction may be more predictable. We therefore propose a new approach of measuring fairness in parallel job scheduling. Our approach seeks to address the weaknesses of the dispersion and fair start time approach identified in Chapter 6. We aim at addressing existing weaknesses that make the current approaches inappropriate.

7.1.2 Addressed weaknesses

We now describe the weaknesses identified in Chapter 6 that we seek to address in our approach to fairness evaluation.

Traffic variation

We know that traffic on supercomputers vary. It involves daily, weekly and monthly peaks which are dictated by the working patterns of the system users. Jobs are likely to have poorer performance during peak hours and better performance during off-peak hours. This should not be represented as unfairness on the side of the scheduler (like done in dispersion). In our approach, a change in traffic does not necessarily imply increase (or reduction) in scheduler unfairness. However, if there is evidence that on top of the poor performance caused by high traffic, the scheduler delays some jobs at the expense of others, then the delays contribute to scheduler unfairness.

Resource requirements

Different jobs have different resource requirements that (partially) determine their schedulability. Resource intensive jobs are mostly un-schedulable and less resource intensive jobs are more schedulable. Comparatively, schedulable jobs perform better than un-schedulable ones. Such a difference in performance is not a result of unfairness. Approaches like dispersion indicate it to be unfairness. It is, however, unfair for the schedulable jobs to have better performance at the expense of the un-schedulable ones. Likewise, it is unfair
the schedulable jobs to be intentionally delayed by the scheduler so as to have the same performance as the un-schedulable ones. Jobs need not to have the same performance to be fairly treated. In our approach, each job is expected to have its individual performance in a fair set up. Such a performance depends on factors like job schedulability and traffic. The deviation from the fair performance of a job is used to measure fairness.

Reference point of performance measurement

In fair start time approach, different jobs have different fair start times. The fair start times for each job depend on the scheduler. This implies that if two schedulers have the same fairness, they do not necessarily have similar levels of discrimination. In dispersion, all jobs have the same point of reference (ideal performance). In our approach, every job is considered to have a fixed (but not necessarily equal) point of reference. The point of reference for each job is independent of the scheduler(s) being studied.

7.2 The Net Benefit Approach

We now describe the net benefit approach to fairness evaluation.

7.2.1 Approach description

Like in other approaches, before we measure fairness, we need an appropriate performance metric. We assume job waiting time to be an appropriate metric though any other metric can be used.

Let us assume we want to compare two schedulers $S_1$ and $S_2$ on fairness. Generally, $S_1$ is fairer than $S_2$ if $S_1$ starves jobs less than $S_2$. For a job to be considered starved, there must be a fair waiting time value (say $w^f$) such that if it waits for more than $w^f$, then it has been unfairly treated (discriminated). If it waits for less than $w^f$, then it has been favored. Let us assume an ideally fair (arbitrary) scheduler $S_f$ which schedules the job stream in such a way that each job gets the fair performance. Let us assume that if the job stream
is scheduled by $S_f$, job $J_i$ has a waiting time $w^f_i$. When scheduled by $S_1$ and $S_2$, $J_i$ has waiting times $w^1_i$ and $w^2_i$ respectively. The net benefit $J_1$ gets when scheduled by $S_1$ and $S_2$ is determined by the extent to which $S_f$ is outperformed by $S_1$ and $S_2$ respectively. If the respective benefits are $b^1_i$ and $b^2_i$ respectively, then

$$b^1_i = w^f_i - w^1_i \text{ and } b^2_i = w^f_i - w^2_i$$

A negative benefit implies discrimination. If $b^1_i > b^2_i$, then $S_1$ is fairer than $S_2$ with respect to $J_i$. Practically, we cannot get the numeric values of $b^1_i$ and $b^2_i$ because we do not know the numerical value of $w^f_i$.

If we are interested in knowing which scheduler, between $S_1$ and $S_2$ is fairer than the other with respect to $J_i$, we compare $b^1_i$ and $b^2_i$. If $b^2_i > b^1_i$, then $S_2$ is fairer than $S_1$ otherwise $S_1$ is fairer than $S_2$. Since $w^f_i$ is unknown, we cannot get the numerical value of $b^1_i$ and $b^2_i$. However, if we compute the difference between the two benefits, we get $b^1_i - b^2_i = w^2_i - w^1_i$ which is independent of $w^f_i$. If $b^1_i - b^2_i < 0$, then $b^1_i < b^2_i$. This implies that to compare the two schedulers for fairness on a certain job, we need not to know the performance of the job when scheduled by a fair scheduler.

If we are to compare say $n$ schedulers, we need $\frac{1}{2}n(n - 1)$ pair wise subtractions for each job which makes the work tedious. To ease it, we can chose a base scheduler (not necessarily the ideal fair scheduler) and compare all other schedulers with it. We then use the relative fairness with the base scheduler to compare schedulers for fairness.

### 7.2.2 Metrics in the approach

To generate the metrics in this approach, let us first generalize benefits generation to the entire job stream. Let us consider a case where a job stream $J_i : i = 1, 2, \ldots N$ is used. We first schedule the job stream by the base scheduler. Let the waiting time for $J_i$ be $w^b_i$. We then schedule the job stream by scheduler $S_k$. Let us assume that the waiting time for $J_i$ is $w^k_i$. The (net) benefit of scheduling the job stream by $S_k$ on $J_i$ is $b^k_i = w^b_i - w^k_i$. 

We compute the benefits for all the jobs in the job stream (a negative benefit implies discrimination).

Let us split the job stream in three (disjoint) sets \( S_0, S_d \) and \( S_b \) made up of jobs with 0, negative and positive benefits respectively.

Let us also define total benefit \( B \) and total discrimination \( D \) as

\[
B = \sum_{J_i \in S_b} b_i^k \quad \text{and} \quad D = \sum_{J_i \in S_d} |b_i^k|
\]

(7.1)

We now define metrics that can be used to measure fairness of \( S_k \)

(i) **Total Discrimination** (\( D \)): This is the total discrimination \( D \) of all discriminated jobs in the job stream.

(ii) **Marginal Discrimination** (\( MD \)): This is the total discrimination in excess of the total benefit in the job stream.

\[
MD = D - B
\]

(7.2)

(iii) **Average Discrimination** (\( AD \)): This is the average discrimination for all discriminated jobs.

\[
AD = \frac{D}{n(S_d)}
\]

(7.3)

(iv) **Extreme Discrimination** (\( D_x \)): This is the total discrimination for jobs in most discriminated proportion \( x \) of the job stream. If \( S_d^x \) is the subset of \( S_d \) containing the most discriminated proportion \( x \) of the job stream, then

\[
D_x = \sum_{J_i \in S_d^x} |b_i^k|
\]

(7.4)

(v) **Average Extreme Discrimination** (\( AD_x \)): This is the average discrimination for all jobs in the most discriminated proportion \( x \) of the job stream.

\[
AD_x = \frac{1}{n(S_d^x)} \sum_{J_i \in S_d^x} |b_i^k|
\]

(7.5)
Extreme Marginal Discrimination ($MD_x$): This is got by getting the marginal discrimination but using the extreme proportion $x$ (of the job stream) for both discriminated and favored jobs. If $S_b^x$ is the subset of $S_b$ containing the most favored proportion $x$ of the job stream, then

$$MD_x = \sum_{J_i \in S_d^x} |b_i^k| - \sum_{J_j \in S_b^x} b_j^k$$  \hspace{1cm} (7.6)

Average Extreme Marginal Discrimination ($AMD_x$): This is the average marginal discrimination for the extreme proportion $x$ of the job stream.

$$AMD_x = \frac{MD_x}{x \times N}$$  \hspace{1cm} (7.7)

### 7.2.3 Example

For illustration, let us consider a job stream of 20 jobs with their benefits tabulated in Table 7.1. Note that jobs are tabulated in the order of increasing benefit which is not necessarily the arrival order.

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_i^k$</td>
<td>-9</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

From the table above, we can observe that:

$$S_d = \{J_1, J_2, J_3, J_4, J_5\}$$

$$S_0 = \{J_6, J_7, J_8, J_9\}$$

$$S_b = \{J_{10}, J_{11}, J_{12}, J_{13}, J_{14}, J_{15}, J_{16}, J_{17}, J_{18}, J_{19}, J_{20}\}$$

to get $B$ and $D$ for the job stream, we add up all the benefits and discriminations (Equation (7.1)). We get

$$B = 1 + 2 + 3 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 58$$
7.3. **EXPERIMENTAL EVALUATION**

\[ D = 9 + 9 + 8 + 7 + 2 = 35 \]

Some of the metrics are for all the jobs. These include \( D \), \( AD \) and \( MD \). We can compute them using the values of \( D \), \( B \) and \( n(S_d) \). We get:

\[ D = 35 \]
\[ MD = 35 - 58 = -23 \]
\[ AD = \frac{35}{5} = 7 \]

Other metrics consider the extremes. We note that much as we can look at the discrimination or marginal discrimination, the proportion used is that of the entire job stream. Therefore, the number of jobs we consider is computed as a proportion of the job stream. The extreme based metric values therefore are:

\[ D_{0.2} = 9 + 9 + 8 + 7 = 33 \]
\[ AD_{0.2} = \frac{33}{4} = 8.25 \]
\[ AD_{0.1} = \frac{9 + 9}{2} = 9 \]
\[ MD_{0.2} = (9 + 9 + 8 + 7) - (10 + 9 + 8 + 7) = -1 \]
\[ MD_{0.1} = (9 + 9) - (10 + 9) = -1 \]
\[ AMD_{0.2} = \frac{-1}{0.2 \times 20} = -\frac{1}{4} \]
\[ AMD_{0.1} = \frac{-1}{0.1 \times 20} = -\frac{1}{2} \]

Different extremes for the same metric have different values. The implications cannot be explained from this example since the values are arbitrary. Detailed implications of the differences are discussed in Section 7.4.2 as we identify the most appropriate metric.

### 7.3 Experimental Evaluation

We now describe an experimental study to evaluate selected schedulers for fairness using our metrics.
7.3.1 Experimental set up

The system

We consider a system of 4 homogeneous clusters of 17 processors each. The clusters are connected by wide-area links which are slower than the intra-cluster links. Co-allocated is possible in the system and co-allocated jobs suffer a 30% execution time penalty. This is to cater for the slower inter-cluster speeds and job communication intensity [7][64]. The system is served by one queue and one scheduler.

Schedulers and placement policy

We use FCFS as the base scheduler. We compare FPFS(5), FPFS(20), Greedy(5,5), Greedy(5,20), Greedy(20,5) and Greedy(20,20) schedulers for fairness. To map components/jobs onto clusters, we use the Worst Fit placement policy.

Job stream

We use traces from the second version of the Distributed ASCI Supercomputer (DAS-2) [82] archived at the Grid Workloads Archive [84]. We make some modifications in the trace. These are:

(i) We only use jobs up to the size of 64. This is done so as to have only jobs that fit in our modeled multi-cluster system (68 processors);

(ii) We remove all jobs that appear repeated. Jobs with the same size, arrival time and execution time are considered duplicated. We only consider one of them in our experiments. This is done to minimize the effect of workload flurries [91].

(iii) Every job whose execution time is 0 is eliminated. The execution time values of 0 came up because job execution times were rounded off to integers prior to archiving. Jobs of execution times less than 0.5 therefore have runtimes of 0 in the traces.
7.3. EXPERIMENTAL EVALUATION

Large jobs, whose size is greater than 10 are broken into 4 components and co-allocated. The co-allocated jobs constitute 9.98% of the jobs and 64.1% of the load.

Measurements

We take measurements between the 7,400th to Job 10,000th jobs (2,600 jobs). This is because within this job range, the system runs at high utilization (Figure 7.1). Using high utilization/load helps reveal performance (and fairness) differences among the job streams.

7.3.2 Results

We compute the fairness of the different schedulers using the different metrics. We present fairness of the different schedulers by $D$ and $MD$ in Figure 7.2. We also present fairness by $AD$ and $MD$ (together with 5% and 10% extremes) in Figure 7.3. We observe that the different measures give different impressions of relative fairness among the schedulers.

For the greedy scheduler, Greedy(5,5) is the most unfair for all the measures. Greedy(20,20) is the most fair scheduler for all the metrics except

![Figure 7.1: Utilization variations in the system](image)
CHAPTER 7. FAIRNESS EVALUATION BY NET BENEFIT

Figure 7.2: Fairness by $D$ and $MD$

$AMD_5$ and $AMD_{10}$ where Greedy(20,5) is the most fair. For the FPFS scheduler, FPFS(20) is fairer than FPFS(5) for $D$, $MD$ and $AD_5$. They are equally as fair for $AD$ and $AD_{10}$ while FPFS(5) is fairer than FPFS(20) for $AMD_5$ and $AMD_{10}$.

Comparison of FPFS and the greedy instances give some contradicting deductions. While Greedy(5,5) is fairer than FPFS(20) and FPFS(5) using AD, the reverse is true for $AD_{10}$ and $AD_5$. Likewise, while Greedy(5,5), Greedy(5,20) and Greedy(20,5) are fairer than FPFS(20) using $D$ but the reverse is true when using $MD$.

The contradictions in relative fairness among the metrics can be attributed to both metric appropriateness and differences in practical implications of fairness for the different schedulers. This implies that the different metrics are actually not all appropriate to measure fairness. To identify which metrics are (not) appropriate, we study the trend of worst performing and most discriminated jobs so as to deduce user satisfaction.
7.4 Validation of the Approach

To validate the approach, we use the trend of the worst performing and the trend of the most discriminated jobs.

7.4.1 Performance and discrimination trends

We now investigate the trend and extent of performance and discrimination of jobs for different schedulers. To do this, we study the trend of the worst 1000 jobs in performance and discrimination. The 1000 jobs are chosen because we assume they are the ones scheduled during peak hours and therefore tell more about the performance/discrimination of the schedulers. Jobs scheduled during off-peak hours are effectively scheduled in their order of arrival (FCFS). They therefore do not represent scheduler characteristics. We summarize the trends of the worst performing and most discriminated jobs in Figure 7.4. From Figure 7.4, we observe that the $i^{th}$ worst performing job for each of the schedulers have different waiting times. The same trend exists for discrimination. We also observe that there are values of $i$ where the relative performance/discrimination between two schedulers change. For example, at $i \approx 40$, the performance of the $i^{th}$ worst job for FPFS(5) and
FPFS(20) is the same. For \( i < 40 \), the FPFS(5) job outperforms FPFS(20) and for \( i > 40 \), FPFS(20) job outperforms FPFS(5). This trend also applies to discrimination.

### 7.4.2 Deductions

We now discuss the trend of performance and discrimination. We then intuitively deduce the appropriate and inappropriate metrics as well as circumstantial cases where specific (appropriate) metrics may be more meaningful.
Performance and discrimination trends

From Figure 7.4, we observe that different schedulers have different trends of performance and discrimination. Performance wise, we observe a fairly smooth reduction in performance with higher rates of change for FPFS schedulers. Discrimination wise, we observe a reduction in discrimination in all schedulers but with the greedy schedulers having some sharp changes. The sharp changes are for schedulers where \( d = 5 \). This implies that the sharp change is largely attributed to the effect of a small \( depth \) value.

The performance trends show that schedulers with \( d = 5 \) give a relatively good performance of worst performing jobs. This is in line with previous studies [62] which showed that at low \( depth \), the greedy scheduler leads to good performance of large jobs. This implies that large jobs get a low discrimination from the greedy scheduler at low \( depth \). However, we observe that Greedy(5,5) has one of the highest discriminations. This implies that the most discriminated jobs in Greedy(5,5) are not necessarily the worst performing jobs. The low \( depth \) actually locks out the small jobs that would otherwise fit in the system (leading to capacity loss). It is the small jobs which miss the scheduling opportunities and hence discriminated by Greedy(5,5). The high discrimination by Greedy(20,5) further confirms this trend. It therefore implies that for Greedy(5,5) and Greedy(20,5), the high discrimination is from small jobs which are not among the worst performing jobs.

Metric appropriateness

Our underlying understanding of fairness is that jobs should not be unnecessarily discriminated. An appropriate metric should not misrepresent discrimination. At the same time, it should not represent a non fair scenario as fair. From Figure 7.4, Greedy(20,20) has the lowest discrimination. At the same time, Greedy(20,20) has the best performance of worst performing jobs. This implies that Greedy(20,20) offers the least discrimination of the jobs. It can therefore be deduced that Greedy(20,20) is the most fair scheduler. From
Figure 7.2 and Figure 7.3, we observe that Greedy(20,20) is the most fair scheduler by all metrics except $AMD_5$ and $AMD_{10}$. We can therefore conclude that $AMD_5$ and $AMD_{10}$ are inappropriate in measuring fairness. This is because they imply unfairness where it is not. On a closer look at Figure 7.2, $MD$ can also be considered inappropriate despite having Greedy(20,20) as the most fair scheduler. The $MD$ metric considers FPFS(20) to be fairer than all schedulers except Greedy(20,20). This is not deducible from Figure 7.4. FPFS(20) has some of the worst performing jobs, the most discriminated jobs. This generally implies that marginal measurements are inappropriate in measuring fairness. This can be attributed to the fact that extra favoritism of jobs leads to a lower marginal value yet favoritism is unfair. In the next section, we only consider non-marginal metrics ($D$, $AD$, and $AD_x$).

### 7.4.3 Circumstantial appropriateness of the metrics

The metrics of $D$, $AD$ and $AD_x$, which have been deduced to appropriately measure fairness, have an approximately similar trend. However, there are some contradictions. These contradictions can depend on the way fairness is envisaged. This can be due to the effect of the numbers and the effect of the extremes.

(i) **The effect of the numbers:**

This is observed in cases where there are metric contradictions between say $D$ and $AD$. One scheduler can have a lower $AD$ but with more jobs discriminated. The relative fairness computed by $AD$ contradicts that of $D$. In our case, it can be observed on FPFS(5) and FPFS(20). Much as FPFS(5) has a higher $D$, they have approximately the same $AD$. Figure 7.4 show that actually FPFS(20) has less discriminated jobs.

(ii) **The effect of the extremes:**

This is observed where there are metric contradictions between say $D$ and $AD_x$. One scheduler may excessively starve the extreme jobs. This leads to a high $AD_x$. However, if it starves a few jobs, then $D$ will be
low. The two, therefore, contradict. Another possibility is having a sizeable number of jobs as well with low discrimination that lowers $AD$ which also causes a contradiction. In our case, it can be observed on FPFS(5) and Greedy(5,5). Due to the many jobs discriminated by FPFS(5), the $D$ value is high. However, the fewer jobs discriminated by Greedy(5,5) have high discriminations. This leads to contradicting $AD_5$ and $AD_{10}$.

In practice, unfairness which is hard to detect by the performance metric is that where a few jobs are extremely discriminated. If the jobs which are highly discriminated are many, then they will substantially affect the average performance metric. It may therefore be more realistic to use $AD_x$ as a generic measure.

The choice of $x$ used is also important. It needs to be low enough to reveal the typical starvation of the extremely affected jobs. Definitely, it is also unrealistic to keep $x$ so low (say 0.001%). It may constitute too little a number of jobs to give a clearer picture of the extremes. It is therefore to the desecration of the system owners that the appropriate value of $x$ is chosen.

## 7.5 Conclusion

In this chapter, we have proposed and evaluated a new approach to fairness evaluation in parallel job scheduling. Our approach considers the net-benefit a job gets by being scheduled by a certain scheduler instead of the base scheduler. We have observed that it caters for differences that can be caused by circumstances of the scheduling environment rather than the scheduler. It therefore reduces the effect of foreign factors like schedulability, job traffic, queue states and system states.

We have proposed different metrics that can be used in this approach. Generally, we have observed that using average extreme discrimination brings out realistic results. It however remains the prerogative of the user to determine the value of the proportion $x$ needs to be small enough to represent the extreme and big enough to indicate a general rather than an isolated trend.
We have validated our approach using the trend of performance and the trend of the discrimination. Our validations show that the metrics can actually represent discrimination. Due to differences in schedulability and scheduler heuristics, we observe that poor performance is not a sole implication of discrimination. Our approach and metrics can be able to distinguish between poor performance caused by discrimination and poor performance caused by circumstances like traffic peaks. More to that, the poorest performing jobs are not necessarily the ones most discriminated. Our approach can single out discrimination done to jobs that are not having the worst performance.
Concluding Remarks

Outline: In this thesis, we have studied techniques of processor co-allocation. We have also studied ways co-allocation can be evaluated. In this chapter, we summarize our work, highlight our contributions and make suggestions for possible future research. In section 8.1, we discuss the overall concept of our research. We then highlight the major contributions of our work in Section 8.2. Finally, we make suggestions for future research in Section 8.3.

8.1 Overview

Parallel job scheduling has been and continues to be an active field of research. On top of parameters being studied, studies differ in terms of the supercomputer platforms, architectures as well as and the mode of processing employed by the supercomputers.

In this study, we have considered a multi-cluster system served by one queue and with one scheduler. The clusters are homogeneous, they process pure space slicing and job migration is not permitted. The jobs are online and rigid with execution times that are unknown until the end of job execution. In case the system considers a certain job to be large enough, it can break it into components and co-allocate it.

Our study started by acknowledging observations made from previous related studies that choosing the metric to evaluate schedulers of computing systems/schedulers have to be done with care. This is because different
metrics can have different implications depending on the circumstances under which they are used. Therefore, there is no universally accepted metric for evaluating parallel job schedulers. Users have to choose the metric that fits best into their environment and use it to evaluate the system.

Even in cases where the metric used appropriately represents performance in a given set up, the average metric value for the entire job stream can be misleading. This is because there are, in some cases, big differences in performance between the best and worst performing jobs. If, for example, the best performing jobs are the majority, then the entire job stream average is highly influenced by them. Deductions from the macroscopic view of performance deductions leave out the details of the discriminated jobs. The deductions, therefore, are not representative of the entire job stream.

In parallel job scheduling, it is rare to have all jobs perform equally. The differences in performance among jobs can be attributed to reasons like:

(i) Some jobs arrive during (traffic) off-peak hours while others arrive in peak hours. Those that arrive in off-peak hours get better performance since they are processed nearly immediately. Those that arrive during peak-hours find a long queue and therefore spend more time in the queue. This leads to poorer performance;

(ii) Some of the jobs, dictated by the resources they require from the system, are more schedulable compared to others. Unschedulable jobs, therefore, spend more time in the queue while schedulable jobs spend less time. This leads to differences in performance;

(iii) The scheduler heuristics are in such a way that they favor some jobs at the expense of others. The jobs that are favored perform better than those discriminated.

The severities of the reasons vary. Reason (i) for example is highly circumstantial. The scheduler has no control of it. Reason (ii) may be looked at as circumstantial as well. However, the scheduler may adjust the severity. This can be done by adjusting some of the job characteristics and improve its
schedulability. In multi-cluster systems, it can be by employing a good partitioning policy. Reason (iii) is more scheduler-based than circumstantial. It largely represents the extent of scheduler unfairness. This can be reduced by improving the scheduler heuristics so that the scheduler is fairer. This thesis has largely addressed reason (ii) and reason (iii). As far as reason (ii) is concerned, we have studied the relationship between jobs characteristics and schedulability, how the characteristics can be modified to improve the overall performance, how communication affects the performance of the scheduler and how differences in schedulability can be used as a base to improve the packing scheme and hence scheduler performance. As far as reason (iii) is concerned, we have studied the concept of favoritism, how favoritism is represented by the existing approaches, weaknesses of the existing approaches. We have also proposed and evaluated a new approach to fairness evaluation.

8.2 Contribution

We now highlight the contribution of our work. We broadly classify the contributions in performance evaluation, fairness evaluation and co-allocation techniques.

8.2.1 Performance Evaluation

Deciding that one system/scheduler outperforms another can be a challenging task. This is because the aggregate value may not adequately represent some of the highly disadvantaged jobs. Much as the disadvantaged jobs are the minority, they have a large proportion of the load. They are therefore not negligible in practice. We therefore studied the performance of the job stream in groups (see Chapter 3) using the FPFS scheduler. We used three parameters to generate the groups. The parameters used were the job size, the number of components and the width of the widest component. We showed that:

(i) There are big differences in performance for the different groups. Gen-
erally, large jobs perform worst while small jobs perform best;

(ii) The job size and the width of the widest component have a direct relationship with job performance/schedulability;

(iii) In cases where a change in parameter leads to improvement in performance, the poor performing jobs get smaller rates of improvement while good performing jobs get a higher rate of improvement;

(iv) In cases where a change in parameter leads to performance deterioration, poor performing jobs get a higher rate of deterioration and good performing jobs get a lower rate of deterioration.

Generally, large jobs perform poorly and they are a minority. Their poor performance makes a negligible effect on the average performance. Nevertheless, they hold the majority of the load. The average performance of the job stream therefore is unrepresentative of the majority of the load.

One common way of improving performance is by appropriately adjusting the job stream/scheduler parameters. However, our studies showed that the large jobs get a comparatively small improvement in performance as parameters are adjusted. This implies that the overall job stream improvement is largely due to the improvement in performance of the small jobs (that make the majority of the job stream). An alternative way of improving the performance is by adjusting parameters that affect job schedulability. In our case where jobs are rigid, size cannot be adjusted but the width of the widest component can. This can be done by employing the phased approach to partitioning. Our studies showed that that the phased approach leads to substantial improvement in performance especially for large jobs.

Performance of co-allocation is highly affected by the effect of communication. In fact, communication may make it unviable to employ co-allocation in multi-cluster systems. Communication effect can be looked at from the job point of view (communication intensity/bandwidth requirements) or from the multi-cluster point of view (intra-cluster to inter-cluster speed ratio/availability of bandwidth). We used a combination of communication intensity and
intra-cluster to inter-cluster speed ratio and investigated the viability of co-allocation (see Chapter 5). We showed that just like in the general performance, using the entire job stream can be misleading. Using the entire job stream can claim co-allocation viability in cases where it is unviable in practice. We also showed that co-allocating very many jobs is not of performance benefit especially when the effect of communication is substantially high. We showed that for every communication penalty value there exists an optimal job size value which is in such a way that if every job bigger than it is co-allocated, we get optimal performance. This value increases as the execution time penalty due to co-allocation increases. We also showed that in case the communication intensity of the jobs in the job stream is not fixed, there is deterioration in performance. Co-allocation is therefore more beneficial if the communication intensity of the jobs co-allocated is fixed. We know of no published work on communication characteristics of super computer workloads. However, due to the differences in sources and applications executed, jobs are very unlikely to have fixed communication characteristics. Co-allocation may therefore not be as viable as implied in studies assuming homogeneous communication.

8.2.2 Fairness Evaluation

Though less studied in parallel job scheduling, fairness remains an important aspect in all queuing systems (including parallel job scheduling). Fairness evaluates the concept of favoritism and discrimination among the jobs. Before schedulers are compared for for fairness, we have to ensure that the metric accurately represent discrimination/favoritism (hence fairness).

We studied the characteristics of common fairness metrics used in previous studies. Specifically, we studied how they represent job discrimination and favoritism in parallel job scheduling set up (see Chapter 6). We examined whether there are cases where a non fair set up can be represented as fair and vice versa. We showed that there are cases where discrimination is implied where it is not (and vice versa). For example, we showed that an increase in
performance dispersion is not always due to discrimination/favoritism. We also showed that a disadvantage in performance on a job caused by another that arrived after it is not necessarily the net disadvantage. Considering it to be a net disadvantage would be partially evaluating the effects of the scheduler heuristics. Broadly, some of the metrics for fairness may be accurate in other queuing systems but have loopholes in parallel job scheduling.

We proposed a new approach to fairness evaluation in parallel job scheduling (see Chapter 7). Our approach uses the net benefit of using one scheduler instead of another to estimate discrimination. It caters for some of the loopholes that exist in the existing approaches. These loopholes include the effect of schedulability, system states, queue states, traffic variations and the long range effects of scheduler decisions. We used the discrimination and performance trends in the job stream to validate our approach/metrics. We showed that using average discrimination, other than the discrimination in excess of benefits gives a more realistic view of fairness. Since unfairness is normally concealed in cases where the affected jobs are few, it is better to use the average of the most discriminated proportion of the job stream. It is to the discretion of the system owners to choose the most appropriate proportion. It needs to be large enough to be representative but small enough to represent the extremes.

8.2.3 Co-allocation Techniques

In performance evaluation, we observed that using the FPFS scheduler, there is a big performance difference between small and large jobs. The performance difference can be attributed to factors like job schedulability and scheduler discrimination. In scheduling techniques, we proposed an approach to minimize the differences between large and small jobs but at the same time improving the overall performance in the system.

We proposed a scheduler that uses the greedy approach to prioritize the jobs (see Chapter 4). In our approach, the priority of the job is a function of its schedulability and the time it has spent in the queue. Our approach gives
8.3. FUTURE RESEARCH

a priority boost to unschedulable jobs. The schedulability boost enhances
the seniority priority used in schedulers like FPFS to give unschedulable jobs
earlier scheduling opportunities. This improves the overall packing since it is
easier to schedule a small job after a large job has been scheduled than vice
versa. Our studies showed that small jobs still outperform large jobs but
with a smaller performance gap. Large jobs registered an improvement in
performance. We also carried out sensitivity studies to the scheduler param-
eters. Sensitivity studies showed big changes in performance for maxJumps
and depth but negligible performance changes for $\alpha$, $\beta$ and $\gamma$. This implies
that if we are to improve performance, less effort need to be put on very
accurate estimation of schedulability. With in some parameter boundaries
(depth and maxJumps), the greedy scheduler outperforms FPFS. Fairness
studies in Chapter 7 also showed that with in some parameter boundaries, the
greedy scheduler is fairer than FPFS. Over all, good results from the greedy
schedulers are got when depth and maxJumps are substantially high.

8.3 Future Research

Our work also opens up more avenues for research in parallel job schedul-
ing. Specifically, more research can be in the effect of alternative estimates
of job schedulability, the effect of communication patterns, communication
cognizant scheduling and fairness evaluation.

8.3.1 Schedulability estimation

Our group wise performance studies showed that there are differences in job
schedulability among the jobs in the job stream. Our greedy scheduler showed
that using (approximate) job schedulability to determine job priority comes
with performance and fairness benefits. However, we used a simple linear
function of a few job characteristics. Extra research can be done on (i) alter-
native approaches to estimate schedulability and (ii) incorporate other job
characteristics (like I/O and communication intensity) in the schedulability
estimation.

Non linear hardness functions can be more accurate and come with better performance since the performance variation with the job parameters (of size, components and width of the widest component) is actually not purely linear. This implies that the benefits got from the linear function can actually be extended. At the same time, in systems that do not process by pure space slicing, other factors like memory requirements and I/O can highly influence schedulability. Incorporating them in the hardness function makes the greedy scheduler also implementable in such set ups.

8.3.2 The effect of communication

In this work, we have looked at only synchronous communication among the jobs. However, we have observed that since the jobs have diverse sources, they do not have a homogeneous communication pattern. We looked at communication heterogeneity in terms of communication intensity. The communication heterogeneity can also be extended to asynchronous communicating jobs. In such a case, we look at job streams that have both synchronously and asynchronously communicating jobs.

Extra research can be done at ways the execution time penalty can be computed for asynchronously communicating jobs as a function of the communication intensity, size, number of components and the intra-cluster to inter-cluster communication ratio. This can be extended to the effect of the job stream composition on the performance and viability of co-allocation.

8.3.3 Communication cognizant scheduling

The viability of co-allocation is highly dependant on the effect of communication. This implies that the schedulers need some improvements in order to reduce the negative effect. Much as our studies show that this can be done by limiting the proportion of the jobs co-allocated, it cab be extended to aspects like selective co-allocation so as to minimize the extra processor hours that come in due to co-allocation. This can be more interesting in job
8.3. FUTURE RESEARCH

streams where both synchronous and asynchronous jobs exist.

8.3.4 Fairness evaluation

Finally, in our net benefit approach, we introduced a base scheduler that is used as a reference point. We used FCFS for simplicity. However, we did not study the effect of the base scheduler on the deductions got from the metrics. There is therefore a need study the effect of the base scheduler on the deductions of the scheduler. There is also a need to investigate which base scheduler best represents the concept of fairness.
Bibliography


Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 2221, pp. 66 – 86.


on Job Scheduling Strategies for Parallel Processing, LNCS 3277, pp. 176 – 194.


on Job Scheduling Strategies for Parallel Processing, LNCS 3834, pp. 283 – 256.


Summary

Computer processing power is increasing at a very high rate. A computer considered to be fast today may not be fast in a few years to come. At the same time, the number and complexity of resource intensive computer applications are increasing. This calls for bringing together of multiple processing units so as to collectively service competing resource intensive applications and employing good scheduling strategies so as to offer the maximum possible satisfaction to the owners of the jobs.

In this thesis, we study the ways rigid jobs can be scheduled on a multi-cluster system that processes by pure space slicing and allows co-allocation. We study ways user satisfaction can be evaluated in a typical multi-cluster system. To a very large extent, this has been done using the average value of the performance metric. Given the nature of typical super computer workloads, jobs have varying resource requirements. This implies that some are more schedulable than others. At the same time, the scheduler may favor some jobs at the expense of others. Studies show that schedulable (small) jobs make up the majority of the jobs but the minority of the load.

Schedulable jobs tend to have good performance while unschedulable ones have poor performance. The good performance of the schedulable jobs (which are the majority) makes the average metric value appear impressive. The impressive average metric value does not imply the poor performance of the majority of the load. We study the differences in performance of the different groups (grouped by size, number of components and width of the widest component) and how the performance varies with the changes in scheduler parameters.

We also study the relationship with job characteristics and their (approximate) schedulability. We show that the schedulability has a big relationship with job size and width of the widest component. We further show that performance can be improved by partitioning the jobs in such a way that they are more schedulable.
Another way we use job schedulability is by using it in prioritization. We use the job (approximate) schedulability to enhance the scheduler prioritization scheme so as to improve the performance of the entire job stream and reduce the performance difference between schedulable and unschedulable jobs. We do this by giving a priority boost to unschedulable jobs on top of the time they have spent in the queue (seniority). We propose the greedy scheduler that uses the new prioritization approach. We show that so long as the depth and maxJumps values are high enough, the greedy scheduler outperforms and is fairer than the FPFS scheduler.

The differences in performance among jobs can be due to differences in job schedulability, cases of the scheduler favoritism/discrimination (unfairness) or a combination of the two. Compared to performance and scheduling techniques, there are fewer studies carried out on fairness in parallel job scheduling. We first study characteristics of existing fairness metrics used in parallel job scheduling. We investigate how they imply fairness/unfairness. We realize that there are instances where the implied unfairness is not unfairness in practice. The deductions can therefore be misleading sometimes. The causes of the misleading deductions are mostly failure to account for the effect of differences in resource requirements for the jobs, differences in job seniority and differences in queue states as the job gets submitted into the queue. We then propose a new approach to fairness evaluation for parallel job schedulers. Our approach considers the job wise net benefit of using one scheduler instead of another. This caters for differences in performance that are not due to scheduler discrimination (like differences in resource requirements and traffic). Broadly, other than comparing a job to others for the same scheduler, our approach compares a job to itself for different schedulers. Our approach addresses the weaknesses found in the existing approaches. We use the performance and discrimination trends to validate our approach on selected multi-cluster schedulers. Our approach is able to deduce unfair treatment of jobs even if the unfairly treated job is not among the worst performing job. Factors like differences in resources among jobs and jobs arriving during peak hours are adequately catered for by our approach as it evaluates scheduler fairness.
Samenvatting (In Dutch)

De kracht van computers neemt in een enorm tempo toe. Wat vandaag nog wordt gezien als een krachtige en snelle computer, wordt morgen al ingehaald door een computer met nog meer processing power. Tegelijkertijd neemt het aantal applicaties met een hoge complexiteit, die dientengevolge veel processing power nodig hebben, ook toe. Deze ontwikkelingen zorgen ervoor dat er een behoefte is ontstaan aan computers waarin in meerdere processoren tegelijkertijd en in collectief ressource intensieve applicaties kunnen bedienen (aangeduid met multi-cluster computer systemen). Daaraan gerelateerd is er een toenemende behoefte aan rekenregels voor het toewijzen van taken (aangeduid met scheduling algoritmen) aan processoren binnen multi-cluster computer systemen. Doelstelling hierbij is om een optimale prestatie te bieden aan de gebruikers van deze computers.

In dit onderzoek gaan we in op de manier waarop rigid jobs gescheduled kunnen worden op multi-cluster computer systemen. We bekijken hoe gebruikerstevredenheid gemodelleerd kan worden voor multi-cluster computer systemen. In de meeste gevallen wordt de gebruikerstevredenheid gemeten aan de hand van de gemiddelde waarde van performance maatstaven. Gegeven de typische kenmerken van werklast verdelingen tussen taken binnen multi-cluster computer systemen, kunnen verschillende taken sterk verschillende werklast behoeften hebben. Dit houdt in dat bepaalde taken makkelijker toe te wijzen zijn dan andere taken. Dit terwijl het scheduling algoritme sommige taken met voorkeur behandelt boven andere taken. Onderzoek heeft aangetoond dat makkelijk toe te wijzen taken vaak de meerderheid van de taken betreft, maar dat zij in totaal de minderheid van de totale werklast vertegenwoordigen.

Makkelijk toe te wijzen taken laten vaak een betere performance zien dan minder makkelijk toe te wijzen taken. De goede performance van makkelijke toe te wijzen taken, die zoals eerder aangeduid vaak de meerderheid vormen, zorgt veelal voor een goede gemiddelde waarde van de performance maatstaf. In dit

143
soort situaties geeft echter een goede gemiddelde waarde geen juist beeld van de zeer slechte performance van de moeilijk toe te wijzen taken, die een minderheid van de taken vormen, maar wel een meerderheid van de werklast representeren. In dit onderzoek onderzoeken wij verschillen in performance tussen verschillende groepen taken (gegroepeerd naar omvang van de taken, aantal componenten in taken, en omvang van de taken met de grootste werklast). Tevens onderzoeken we hoe de performance van scheduling algoritmen varieert bij het variren van parameters van de scheduling algoritmen.

Wij onderzoeken de manier waarop specifieke karakteristieken van taken invloed hebben op de mate van gemak waarmee taken toegewezen kunnen worden aan processoren. We laten zien dat de mate van gemak van toewijsbareheid van taken een sterke relatie vertoont met de grootte van een taak en de omvang van de taak met de grootste werklast. We laten ook zien dat overall performance bevoeld wordt door het opdelen van taken in deeltaken, zodat de mate van gemak van toewijsbareheid toeneemt. Daarnaast laten we zien dat de mate van gemak van toewijsbareheid gebruikt kan worden in het prioriseren van taken. We gebruiken de mate van gemak van toewijsbareheid voor het verbeteren van toewijzingsmechanismen in scheduling algoritmen, met als doel de performance van alle taken te verbeteren en het verschil in performance tussen makkelijk en moeilijk toe te wijzen taken te verminderen. We doen dit door het introduceren van een nieuw scheduling algoritme, genaamd de greedy scheduler, waarin de prioriteit van taken in de wachtrij berekend wordt door een combinatie van de mate van gemak van toewijsbareheid van een taak en de tijd die de taak al in de wachtrij heeft doorgebracht om toegewezen te worden (aangeduid met senioriteit). We laten zien dat, onder bepaalde voorwaarden, de greedy scheduler een betere scheduler is dan bestaande scheduling algoritmen zoals het Fit Processors First Served (FPFS) algoritme.

Het verschil in performance tussen taken kan veroorzaakt worden door verschillen in gemak van toewijsbareheid van specifieke taken, situaties waarin het scheduling algoritme bepaalde taken een voorkeursbehandeling geeft boven andere taken (aangeduid met het begrip eerlijkheid) of een combinatie van deze twee. Recent onderzoek richt zich met name op het ontwikkelen van scheduling algoritmen voor multi-cluster computer systemen en de performance van deze algoritmen. Het begrip eerlijkheid krijgt in de literatuur heel weinig aandacht. Wij onderzoeken karakteristieken van bestaande maatstaven voor eerlijkheid voor scheduling algo-
ritmen en in hoeverre deze karakteristieken echt een uitspraak doen over gerealiseerde eerlijkheid in het toewijzen van taken aan processoren. We zien dat in sommige gevallen deze maatstaven eerlijkheid pretenderen, terwijl dit in werkelijkheid niet het geval is. Gevolgtrekkingen aan de hand van deze maatstaven blijken in de praktijk misleidend te zijn. De redenen voor misleidende gevolgtrekkingen blijken te liggen in verschillen in omvang van taken, verschillen in senioriteit van taken en verschillen in de status die taken hebben indien ze in de wachtrij staan om toegewezen te worden aan processoren. Wij introduceren een nieuwe maatstaf voor het meten van het begrip eerlijkheid. Onze aanpak is gebaseerd op een techniek waarin wordt gekeken naar het verschil in netto toegevoegde waarde voor taken tussen verschillende scheduling algoritmen. Dit betekent dat we, in plaats van een vergelijking in performance van verschillende taken bij n scheduling algoritme, hier voor n taak verschillende scheduling algoritmen met elkaar vergelijken. We valideren onze aanpak op geselecteerde multi-cluster computer systemen. Onze aanpak geeft de zwakheid in de eerder ontwikkelde maatstaven weer, en geeft aan dat de door ons ontwikkelde maatstaf een veel betere waardering geeft voor het begrip eerlijkheid. Onze aanpak is in staat om oneerlijke behandeld taken te ontdekken, zelfs indien de oneerlijk behandelde taak niet n van de slecht presterende taken is. Factoren zoals verschillen in omvang van taken en de tijd wanneer taken in het systeem verschijnen tijdens bijvoorbeeld piekuren in de werklast, worden adequaat gesignaleerd en verwerkt in het realiseren van een eerlijk toewijzingsmechanisme.
Curriculum vitae


In November 2004, John joined Radboud University Nijmegen as a junior researcher with the express intent of obtaining a Ph.D. Currently, John is an Assistant Lecturer in the Department of Computer Science, Faculty of Computing and Information Technology, Makerere University.