Workload modeling and performance evaluation in parallel systems

Minh, T.N.

Workload Modeling and Performance Evaluation in Parallel Systems

Trần Ngọc Minh

Workload Modeling and Performance Evaluation in Parallel Systems

PROEFSCHRIFT

ter verkrijging van

de graad van Doctor aan de Universiteit Leiden,

op gezag van Rector Magnificus prof. mr. P.F. van der Heijden,

volgens besluit van het College voor Promoties

te verdedigen op donderdag 22 maart 2012

te klokke 15.00 uur

door

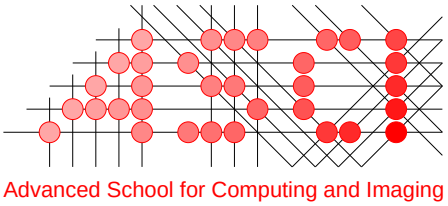
Trần Ngọc Minh

geboren te Dong Nai, Vietnam in 1982

Promotiecommissie

Promotoren:	Prof.dr. H.A.G. Wijshoff
Co-promotor:	Dr. A.A. Wolters
Overige leden:	Prof.dr. D.H.J. Epema (Delft University of Technology)
	Prof.dr. K.A. Gallivan (Florida State University, USA)
	Prof.dr. F.J. Peters
	Prof.dr. J.N. Kok

This work was funded by NWO in the project Guaranteed Delivery in Grids.



This work was carried out in the ASCI graduate school.
ASCI dissertation series number 255.

Workload Modeling and Performance Evaluation in Parallel Systems
Trần Ngọc Minh
Thesis Universiteit Leiden
ISBN: 978-90-8891-387-7
Printed by: Proefschriftmaken.nl
Published by: Uitgeverij BOXPress, Oisterwijk

To my mother and father

To my younger sister, my son and my wife

Contents

1 Introduction **1**

1.1 Research Statement 2

1.1.1 Scheduling Problem 3

1.1.2 Two Challenges in Scheduling Design 3

1.1.3 Challenge from Potential Impacts of Workloads 4

1.2 Related Work 5

1.3 Thesis Organization 7

1.3.1 Chapter Overview 7

1.3.2 List of Publications 10

1.3.3 Roadmap 11

2 Background Knowledge **13**

2.1 Background Statistics 13

2.1.1 Marginal Statistics 13

2.1.2 Autocorrelation 14

2.1.3 Cross-correlation 14

2.2 Representations of Arrival Events 15

2.3 Definitions of Workload Features 16

2.3.1 Features of an Arrival Process 17

2.3.2 Features of Runtime and Parallelism 19

2.3.3 The Bag-of-Tasks Behaviour 22

2.4 Summary 23

3 Statistical Analysis **25**

3.1 Workload Data under Study 25

3.2 Job Arrival Analysis 27

3.3 Runtime and Parallelism 29

3.3.1 Runtime Classification 29

3.3.2 Examination of Correlation 30

3.3.3 Examination of Temporal Locality 30

3.3.4 Examination of Spatial Burstiness 33

3.4 Basic Analysis of Bag-of-Tasks 33

3.5 BoT Arrival Analysis 35

3.5.1 Interarrival Times and Temporal Burstiness 36

3.5.2 Long Range Dependence and Periodicity 37

3.6 BoT Size, Runtime, Parallelism and Estimate 40

3.7 BoT Performance 43

3.8 Importance of Workload Features 44

3.9 Summary 45

4 Modeling Job Arrivals **47**

4.1 Theory of Multifractal Wavelet Model 47

4.2 MWM and Burstiness 49

4.3 Data Modeling and Synthesis of Standard MWM 50

4.4 Modification of MWM 51

4.5 Experimental Results 54

4.5.1 Long Range Dependence 55

4.5.2 Temporal Burstiness 55

4.6 Summary 56

5 A Comprehensive Parallel Workload Model 57

5.1 The Comprehensive Model 57

5.1.1 Runtime Classification 57

5.1.2 Performance of BoTs 58

5.1.3 Parallelism Classification 58

5.1.4 The Complete Model 59

5.2 Experimental Results 61

5.2.1 Bag-of-Tasks Behaviour 62

5.2.2 Spatial Burstiness 62

5.2.3 Marginal Distributions 63

5.2.4 Correlation between Runtime and Parallelism 64

5.2.5 Simulation-Based Evaluation 64

5.3 Related Work 65

5.4 Summary 66

6 Performance Impact of Job Arrivals on Clusters and Grids 69

6.1 Workload Models 69

6.2 Realistic Simulation and Requirements 70

6.3 System Scenarios 71

6.4 Experimental Results 71

6.4.1 Cluster Scenario 72

6.4.2 Grid Scenario 74

6.5 Related Work 76

6.6 Summary 77

7 Performance Impact of Runtime and Parallelism on Scheduling 79

7.1 Control Correlation 79

7.2 Realistic Simulation Methodology 84

7.2.1 Applied Workload Model 84

7.2.2 Scheduling Policies 84

7.3 Experimental Results 85

7.3.1 Performance Impact of Correlation 85

7.3.2 Performance Impact of Locality 89

7.4 Discussions 89

7.5 Summary 91

8 A History-Based Predictor of Parallel Application Runtimes 93

8.1 Related Work 94

8.2 Workloads Under Study 95

8.3 Predict Application Runtime 96

8.3.1 Define Job Similarity 96

8.3.2 Predict the Runtime for a Future Job 98

8.3.3 Train for Best Parameters 100

8.4 Experimental Results 103

8.4.1 Metrics Used in Evaluation 103

8.4.2 Underestimation Problem 104

8.4.3 The Mean Absolute Error 105

8.4.4 The Weighted Absolute Error 106

8.5 Summary 107

9 Conclusions 109

Samenvatting 123

Acknowledgements 125

Curriculum Vitae 127

Chapter 1

Introduction

Parallel processing is the key to fulfill high demands in large-scale computing nowadays. Not only scientific but also industrial applications have become larger and inexecutable on a sequential computer due to time-constraint requirements. This has further pushed research in parallel computing into the mainstream, where parallel systems play a central role. They help to shorten the execution time of large applications, so increase user performance. Furthermore, the role of parallel systems is more important with the emergence and the rapid evolvement of grid and cloud computing because they are used as underlying basic components in grids and clouds. The fact is that parallel systems are growing rapidly in size, showing the urgent need of large-scale parallel platforms. Using data obtained from the TOP500 Supercomputing Sites [103], Figure 1.1(a) indicates that parallel systems with less than 8K processors seem unable to satisfy user demands in recent years and so larger parallel systems have been developed as a consequence as shown in Figure 1.1(b). Along with the rapid growth of parallel systems, questions regarding their performance like "how to design an efficient parallel scheduler?" are always challenging issues. Although research on enhancing the performance of parallel systems through smart scheduling designs has grown since numerous decades, parallel job scheduling is still a hard research topic and far from obtaining an efficient solution in practice. Investigating the Parallel Workloads Archive [80], we see that most parallel systems nowadays still implement the simple scheduling policy First-Come-First-Served with the support of backfilling despite several new scheduling policies are proposed annually [27]. Therefore, we believe that research on parallel scheduling should continue actively to bring efficient solutions into practice.

One of the key factors that has a strong effect on the scheduling performance of a system is the workload. Based on workloads, there are four steps that should be made to obtain an effective scheduling design. Firstly, a careful analysis of real workloads (also called traces) is essential to achieve a good understanding of job nature or user behaviour. Secondly, real traces with a limited availability are not enough to assess the quality of a scheduling algorithm because hundreds of simulations with hundreds of different workloads need to be investigated to ensure the accuracy of the results. Hence, statistical workload models that can capture characteristics observed from the

analysis should be created. The models can help to generate as many workloads as desired. Thirdly, workload models are used in experiments to evaluate the performance impacts of realistic observed workload characteristics on schedulers. Through the experiments, a better understanding of workload characteristics is derived and useful clues can be constructed to take advantage of the characteristics for better scheduling decisions. Finally, with a good understanding of real workloads and the achieved clues, efficient scheduling policies can be designed and ideally evaluated by means of simulation before they are implemented in real systems.

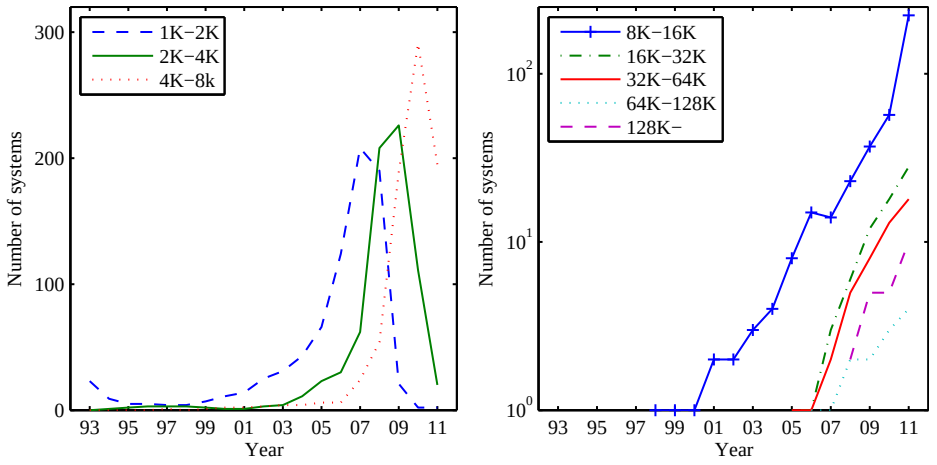


FIGURE 1.1: The growth of parallel systems in terms of number of processors. Data are obtained from the TOP500 Supercomputing Sites [103].

This thesis focuses on the first three steps while the last step is still an open question. The core of the thesis is to fully exploit workload data for performance evaluation of parallel systems. The following sections introduce the main research problem, present related work and end with the organization of the thesis.

1.1 Research Statement

Enhancing the performance of parallel systems needs research efforts on scheduling designs because job scheduling is a fundamental operation on these systems. This section introduces the problem of parallel system scheduling and non-trivial challenges that will be addressed in this thesis.

1.1.1 Scheduling Problem

The term "scheduling" has a broad variety of meanings in parallel system performance, depending on the perspective for optimization. From the perspective of a user, it can be considered as application/task scheduling [43, 74] where the objective is to find the best schedule for tasks of a single parallel application submitted by the user. However, this kind of scheduling does not take care of other applications running on the same system, so can potentially harm the whole system performance. This research concerns primarily with job scheduling, where the purpose is to decide when and where each job should be executed from the perspective of the system [111]. The term "job" refers to some application which can be either serial or parallel. The scope of this study is for rigid jobs which do not change their requested number of processors at runtime. A job consists of three attributes: the arrival time indicating the submission time of the job, the runtime indicating how long the job is executed, and the parallelism indicating how many processors the job requests. Once a job is allocated to processors, it can exclusively use the processors until its execution is complete.

The job scheduling problem in parallel systems can be described as follows. *Given a parallel system consisting of N processors and a parallel workload consisting of M jobs, and assume that all of these jobs are submitted to the system for execution according to their arrival time. The scheduling purpose then is to find an efficient schedule to allocate M jobs to N processors so that the system can obtain the maximal utilization.*

There are several non-trivial challenges in the field of parallel job scheduling [26]. This thesis does not tackle directly the scheduling problem but focuses on the key challenges to reach possible optimal solution for the problem. These challenges are raised from either the designed scheduling strategies or the designers, and are described in Sections 1.1.2 and 1.1.3, respectively.

1.1.2 Two Challenges in Scheduling Design

Once a new scheduler is designed, its efficiency should be ideally evaluated before it is implemented in a real system. This is crucial because the expensive cost of the real implementation will become wasteful if the evaluation is inexact. Currently, performance evaluation work is done mainly by simulation, which requires representative workloads to produce reliable results. There are two kinds of workloads typically used in scheduling evaluation: real workloads (traces) collected from real systems, and synthetic workloads generated by statistical models. The advantage of using a trace directly is that it reflects precisely a system in practice. However, the drawback is that the trace is only a single specific workload collected in the past. Even when a scheduler is evaluated to work well with a trace, there is always the question of whether the evaluation result is general. This means it is not sure if the scheduler still works

well with a trace in the future since a workload can and will evolve over time. The design of a new scheduler should be evaluated by means of simulation with hundreds of workloads to ensure good scheduling performance with future workloads. This fact leads to the urgent need of representative workload models that help to generate several synthetic workloads to fulfill the requirement of the simulations. Therefore, the major challenge for scheduling evaluation is the creation of representative workload models.

In order for a good scheduling decision, useful information must be transferred to the scheduler. It is crucial that the transferred information is of high quality. One of the most important scheduling information is the runtime of a job that is estimated before execution. This information is particularly useful for backfilling scheduling policies, which are implemented intensively in current parallel systems. The estimated runtime of a job can be either provided by its user or predicted automatically by the system. However, user estimated runtime is relatively inaccurate [73] and so system predictors become more favorite [70]. Hence, another challenge for scheduling design is the necessity of an efficient runtime predictor.

1.1.3 Challenge from Potential Impacts of Workloads

The two challenges described in Section 1.1.2 are related to the central role of workloads, which leads to the next challenge in finding an efficient solution for the scheduling problem. It is essential for scheduling designers to have a good understanding of real workloads before they start designing scheduling strategies. Towards this end, several long-term parallel workloads have been collected [80] and characterized [19, 48, 51, 63]. From these analysis studies, a number of workload features[1] have been found, namely *periodicity*, *long range dependence*, *temporal burstiness* of job arrivals, *Bag-of-Tasks* behaviour, *spatial burstiness*, *temporal locality* and *correlation*. The *periodicity* is observed since jobs are often submitted in cycles, in particular daily cycles. The *long range dependence* in a job arrival process means that successive samples of the process are not independent of each other, instead they are correlated. The notion of *temporal burstiness* is the tendency of arrivals to occur in bursts, separated by long periods of no arrivals. The *Bag-of-Tasks* behaviour refers to the submission of several similar jobs within a small duration. The *spatial burstiness* of a parallel workload refers to the non-uniformity of the distributions of runtimes and numbers of processors. The phenomenon of *temporal locality* is understood as a persistent similarity between runtimes of consecutive jobs. The *correlation* feature refers to the cross-correlation between numbers of processors and runtimes. We refer to Chapter 2 for formal definitions of these characteristics. These workload features may potentially have significant impacts on scheduling performance. A deep understanding of these features will be helpful to find useful clues for enhancing parallel schedulers.

[1] The words "feature", "property" and "characteristic" will be used interchangably with the same meaning throughout this thesis.

Therefore, the challenge question with regard to workloads that will be studied in this thesis is *"What are the performance impacts of workload features on parallel system scheduling?"*.

1.2 Related Work

The main objective of a workload model is representativeness, i.e., how well the model represents real workloads. The representativeness depends on the modeling approach. The simplest approach is to use naive models to generate workloads such as sampling values from Poisson or uniform distributions. These models are not based on real workloads and so their representativeness is quite low. However, several scheduling studies do opt to use naive models [2, 109, 112], possibly due to a lack of realistic models. A better approach that is often used in workload modeling is to find the best distributions fitting real data, and, then to sample these distributions to obtain synthetic data. This approach is simple in that it is easy to find the best fitted distribution via the Maximum Likelihood Estimation (MLE) method [76] and Goodness-of-Fit (GoF) tests [19]. The representativeness of this approach is better than that of a naive model because it takes into account the distributions of real data. Studies in [8, 40] are based on this approach. In [8], Cirne and Berman model job arrivals by numerically fitting a polynomial to the job arrival rate, job runtimes with the gamma and the uniform-log distributions, and job parallelisms with the uniform-log distribution. Jann et al. [40] model these job attributes with distributions such as the exponential, the hyper exponential, the Phase Type, and the Erlang and Hyper Erlang Distribution of Common Order. However, we argue that the representativeness still can be much improved since this approach often neglects many realistic properties observed in real workloads. The best modeling approach focuses on capturing realistic workload properties. When investigating workload models in the literature that are developed with this approach, we see that capturing a workload property well also helps for a good fit of marginal distributions. Therefore, this approach is more representative than the two modeling approaches mentioned above and it becomes a favorite choice of numerous modeling studies [18, 39, 59, 95]. The problem with this approach is that there are no standard tools like the MLE method and GoF tests as in the second approach. In [39], Iosup et al. model the Bag-of-Tasks (BoT) behaviour for grid workloads. The authors assume that all jobs are serial and so their model is not suitable for parallel jobs. In [18], Feitelson has proposed a general repetition approach to model locality in job runtimes. However, his approach requires an efficient method to determine the similarity between runtimes so that we can know when a runtime is similarly repeated. Such a method is suggested in [54], which we use and expand to model not only temporal locality but also BoTs. Song et al. [95] develop a parallel workload model based on Markov chains [41] to capture the cross-correlation between runtimes and parallelisms while neglecting the other properties. The cross-correlation feature is also modeled by Lublin and Feitelson [59], where the daily cycle characteristic is included. Indeed, it is not easy to capture a workload property be-

cause it needs a profound analysis and understanding of real world data. It is even more difficult if one wants to incorporate several workload properties into a model simultaneously. Current models [8, 18, 39, 40, 59, 95] can only capture from zero to at most two workload characteristics. As a matter of course, the more characteristics a model can capture, the more representative it is. This is also the advantage of our study since we provide a realistic model that incorporates up to five important workload properties at the same time, as is introduced and elaborated later in Section 1.3.1 and Chapter 5.

With respect to research on performance evaluation, though several workload features are considered to be present commonly in real data and they certainly have potential impacts on scheduling, these performance impacts are hardly described in the literature. We know about only four performance studies on workload features [22, 39, 46, 57]. In [39] performance issues of the Bag-of-Tasks behavior are described. The scheduling effect of daily cycles of activity is investigated in [22]. The correlation feature is studied in [57] while long range dependence is researched in [46]. It should be noted that these studies only focus on the performance of single workload feature with an assumption about the absence of the other ones. Therefore, the question of *"What are the combined impacts of workload features on scheduling performance?"* is still open. Furthermore, the performance of several workload properties such as temporal burstiness and temporal locality is still not known.

The work presented in this thesis builds on and expands the research of Li [47], which also focuses on workload modeling and performance evaluation. With regard to workload modeling, he developed separate workload models for the arrival time and the runtime of grid jobs, which was a significant contribution to the grid research community. His models do not take the parallelism into account because most grid jobs in practice are serial. While his models can be efficiently applied to grids, scheduling research in parallel systems requires parallel workload models which must contain the parallelism attribute. It is difficult to model the parallelism because this job attribute is demonstrated to have a correlation with the runtime attribute [19]. Therefore, it is necessary to develop a new workload model for parallel jobs, which is an important content of this thesis. We start to develop our parallel workload model by expanding his job arrival model. In [47], he introduced a model for job arrivals with the long range dependence and we expanded this model by modifying it to include the temporal burstiness, besides the long range dependence. Furthermore, we developed the expanded model to a comprehensive model with not only job arrivals, but also job runtimes and job parallelisms. In addition to the long range dependence and the temporal burstiness, the comprehensive model can also capture the spatial burstiness, the Bag-of-Tasks behaviour and the cross-correlation between runtimes and parallelisms. For performance evaluation, Li contributed a significant study on the scheduling performance of the autocorrelation feature in grid workloads. Although this enriches the understanding of an important workload characteristic, the performance of other workload features should also be investigated since they may have potential significant impacts on scheduling. Hence, we expand his work with a study of the impacts of the long range dependence, the temporal burstiness,

the temporal locality and the cross-correlation between runtimes and parallelisms as well as the performance interaction between them.

1.3 Thesis Organization

This thesis consists of nine chapters. Beside Chapter 1 presenting the introduction, the other eight chapters are described in this section. In addition, we provide a list of publications that are connected to each chapter and a roadmap to instruct the reader an easy way to read the thesis.

1.3.1 Chapter Overview

The following paragraphs summarize the remaining chapters of this thesis:

Chapter 2 introduces necessary background knowledge and technical terms used in this thesis. The provided background knowledge includes the definition and the different representations of arrival events. Basic statistical and probability measures are presented such as distribution functions, first order statistics and correlation functions. In addition, several important features of a point process such as *periodicity*, *long range dependence*, *temporal* and *spatial burstiness*, *temporal locality*, *(auto/cross)-correlation*, etc. are also presented. These features play a crucial role where the research as described in this thesis centers around.

Chapter 3 firstly presents the real world data used in this research and the details of the real systems where the data are collected. Then the chapter analyzes the real workloads with respect to the statistical features introduced in Chapter 2. The analyzed data cover not only parallel but also grid traces because both kinds of data are similar in many situations. Hence, research results in later chapters can also be applied for grids. The analysis is done statistically with two motivations. Firstly, it is essential to show the common presence of the statistical features in real world data and thus, push the urgency of research on discovering the potential impacts of the features on scheduling performance. Secondly, the empirical analysis of a parallel system results in a statistical overview on how the system is used in practice. This increases the understanding of the system and so helps to make better scheduling designs. In particular, this chapter makes a large focus on the analysis of Bag-of-Tasks behaviour. This behaviour has recently drawn the attention of the scheduling community and seems to be common in workloads of parallel systems [71] and grids [38]. Therefore, the characterisation of the Bag-of-Tasks behaviour helps further improve the understanding of this well-known behaviour in parallel system workloads. The characterisation will focus on the statistical features of Bag-of-Tasks attributes. In addition, we also analyze the performance of Bags-of-Tasks to show how they consume computing resources in parallel systems.

Chapter 4 proposes a model for job arrivals, which is an essential part of parallel workload modeling. Although a job arrival process has many important characteristics such as long range dependence and temporal burstiness, in many studies, it is assumed to be a Poisson process in evaluation work, for simplicity. Since these two characteristics may potentially affect the performance evaluation process, we argue that any job arrival model should take them into account to make itself more realistic. Despite the fact that there are several studies which model long range dependence for an arrival process, most of them focus only on network traffic and less on system job traffic. Furthermore, to our knowledge, there is currently almost no research focusing on both long range dependence and temporal burstiness at the same time. With respect to this research trend, the multifractal wavelet model [85] recently has been introduced as a good choice to yield long range dependence for a job arrival process. Though long range dependence is well captured, we observe that a job arrival process produced by this model does not keep temporal burstiness. In this chapter, we present our study on modifying the multifractal wavelet model to adapt it to system job traffic so that not only long range dependence but also temporal burstiness are produced in a synthetic job arrival process that is generated by the adapted model.

Chapter 5 suggests a comprehensive realistic model for parallel system workloads, which is developed from the job arrival model proposed in Chapter 4. A job generated by this comprehensive model consists of three attributes, namely the arrival time, the runtime and the parallelism. Several statistical studies [18, 19, 48, 51, 56] have shown that the characteristics of parallel system workloads are far from independently distributed. Instead, they have numerous important and correlated characteristics such as the cross-correlation between runtime and parallelism, spatial burstiness, Bag-of-Tasks behaviour, etc. All of these properties may have significant impacts on the scheduling performance of parallel systems. In fact, there are many studies on providing workload models to capture these properties [19, 39, 45, 59]. However, these studies only capture each characteristic separately and ignore others. Therefore, using these models in evaluating scheduling algorithms will lead to potentially inaccurate results because the impact of the interactions among the characteristics are neglected. Hence, we argue that a workload model should incorporate as many realistic workload properties as possible. The comprehensive and realistic model we propose in this chapter will concurrently capture several important characteristics of parallel system workloads including 1) long range dependence, 2) temporal burstiness, 3) spatial burstiness, 4) Bag-of-Tasks behaviour, and 5) correlation between runtime and number of processors. This model is comprehensive and highly realistic, compared with current workload models in the literature, because it can capture up to five realistic workload features at the same time and covers three workload attributes that are essential for simulation of schedulers.

Chapter 6 studies how long range dependence and temporal burstiness structures of job arrivals in a workload affect the performance of clusters and grids. With this study, we would like to deepen the understanding of the scheduling performance impacts of long range dependence and temporal burstiness. The study uses the representative model proposed in Chapter 4 to generate job arrivals with these charac-

teristics to drive simulations of realistic system scenarios. Our experiments show that background workloads with long range dependence and temporal burstiness structures cause a performance degradation of a cluster but seem to have little impact on a grid. In contrast, grid workloads with these properties do not affect the scheduling performance of a cluster but decrease that of a grid. Furthermore, we also indicate that using workloads with/without these features in evaluating cluster schedulers may produce different results.

Chapter 7 investigates the scheduling performance of parallel systems under various patterns of parallel workloads, namely those with the two features of temporal locality and correlation between runtime and parallelism. Feitelson [18] recently indicated that temporal locality is a newly identified phenomenon in parallel workloads and it is essential to investigate its effects on parallel systems. According to our knowledge, there are few studies on the impacts of the temporal locality and correlation features on scheduling performance. Since these impacts are not well known, scheduling algorithms are often evaluated with random workloads, which neglect the phenomenon of temporal locality and the correlation. Because our study shows that these two features can significantly affect the performance of schedulers, this can result in an inaccurate scheduling evaluation for parallel systems. In our simulation-based experiments, an increase of the correlation can quickly degrade the parallel system performance and can change the result of comparing different scheduling policies. With respect to temporal locality, we indicate that this feature does not always seriously affect schedulers of parallel systems, but in particular situations, it can help to improve scheduling performance. Furthermore, we also discuss in this chapter the necessity of using workloads with these features in scheduling evaluation as well as how to utilize the features to enhance the performance of schedulers.

Chapter 8 presents a novel method to predict job runtimes on backfilling parallel systems. Backfilling algorithms [73] have become more popular for scheduling in many space-shared computing resources nowadays. With backfilling, an estimate of job runtime is necessary for each job to determine whether it is short enough to be backfilled. The estimate can be either provided by the user or predicted automatically by the system. However, the user estimated runtime is found to be extremely inaccurate [73] and therefore efficient system predictors for job runtimes should be developed. The predictor proposed in this chapter is based on mining historical data to obtain important parameters. These parameters are then applied to predict the runtime of future applications. It has been shown in previous works [73, 108] that both underestimation and inaccuracy in prediction have adverse impacts on scheduling performance of backfilling systems. In this study, we try to reduce the number of jobs that are underestimated and reduce the prediction error as much as possible. Comparing with other predictors, experimental results show that our predictor is upto 25% better with respect to the problem of underestimation. Moreover, using the metric proposed in [108] for the accuracy, our predictor improves upto 32%.

Chapter 9 summarizes the whole thesis with conclusions about workload characterisation, workload modeling and performance evaluation in parallel systems and

grids. The chapter finalises this thesis by suggesting some open questions for future research.

1.3.2 List of Publications

The following list of peer-reviewed papers are connected with research chapters in this thesis:

Chapter 3

- T. N. Minh, L. Wolters, "Towards a Profound Analysis of Bags-of-Tasks in Parallel Systems and Their Performance Impact", International ACM Symposium on High Performance Parallel and Distributed Computing (HPDC), pp. 111-122, 2011.

Chapter 4

- T. N. Minh, L. Wolters, "Modeling Job Arrival Process with Long Range Dependence and Burstiness Characteristics", International Symposium on Cluster Computing and the Grid (CCGRID), pp. 324-330, 2009.

Chapter 5

- T. N. Minh, L. Wolters, D. Epema, "A Realistic Integrated Model of Parallel System Workloads", International Conference on Cluster, Cloud and Grid Computing (CCGRID), pp. 464-473, 2010.
- T. N. Minh, L. Wolters, D. Epema, "A Systematic Approach for Parallel Workload Modeling", Annual Conference of the Advance School for Computing and Imaging (ASCI), 2010.
- T. N. Minh, L. Wolters, "Modeling Parallel System Workloads with Temporal Locality", Job Scheduling Strategies for Parallel Processing (JSSPP), LNCS, vol. 5798, pp. 101-115, 2009.

Chapter 6

- T. N. Minh, L. Wolters, "Performance Impact of Job Arrivals on Clusters and Grids through Realistic Model-Based Simulation", International Symposium on Performance Evaluation of Computer & Telecommunication Systems (SPECTS), pp. 22-29, 2011.

Chapter 7

- T. N. Minh, L. Wolters, "Model-Driven Simulation to Evaluate Performance Impact of Workload Features on Parallel Systems", International Conference on Cluster Computing (CLUSTER), pp. 84-92, 2011.

Chapter 8

- T. N. Minh, L. Wolters, "Using Historical Data to Predict Application Runtimes on Backfilling Parallel Systems", International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 246-252, 2010.

In addition to the above publications, a journal paper has been submitted:

- T. N. Minh, D. Epema, L. Wolters, "Analysis and Modeling of Parallel Workloads with Realistic Characteristics", IEEE Transactions on Parallel and Distributed Systems (TPDS), 2011. (submitted)

1.3.3 Roadmap

Figure 1.2 draws a roadmap to help the reader easily follow this thesis. We recommend that before reading a chapter, the reader should have a look at its preceding chapter, which provides necessary knowledge and information for the reading.

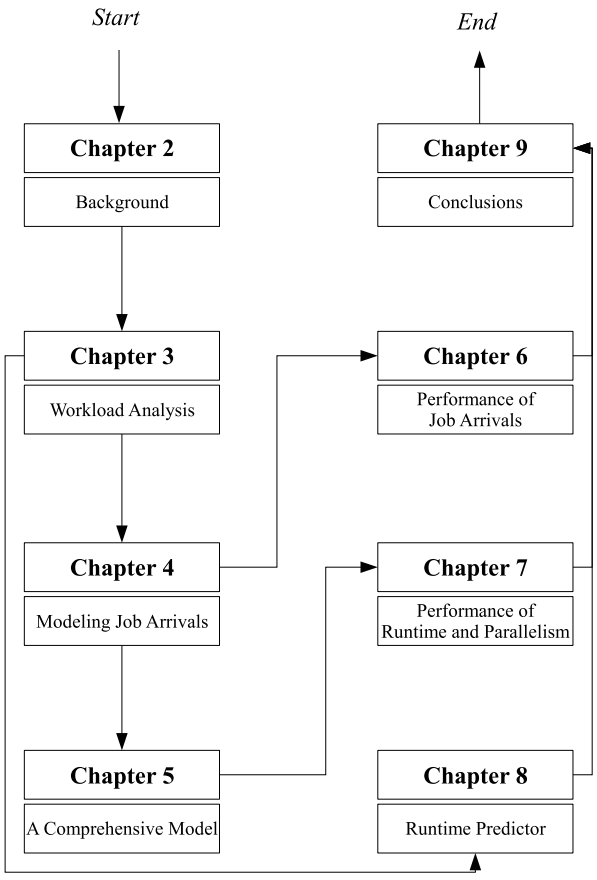


FIGURE 1.2: Roadmap of reading the thesis.

Chapter 2

Background Knowledge

This chapter provides basic theories used in workload characterisation and modeling such as statistical knowledge and methodologies. Throughout this thesis, the chapter should be the first choice of reference for the reader, where further references will be given for deeper understanding. The chapter starts with basic statistical measures such as distribution functions, first order statistics and correlation functions. Then, different representations of arrival events are introduced. Finally, definitions of important characteristics of point processes are presented, which play a central role in the research target of later chapters. They serve as research methodologies in workload analysis and as the objectives that workload models should capture.

2.1 Background Statistics

This section covers some background concepts used in the statistical analysis of point processes. Since these concepts are presented in brief, we refer readers to [4, 19, 47, 86] for more information about these basic measures.

2.1.1 Marginal Statistics

Let $X = \{X_n\}$ be a stochastic point process. We present in this section some marginal properties of X, including mean μ, variance σ^2, standard deviation σ, coefficient of variation C_v, cumulative distribution function (CDF) $F(x)$, and complementary cumulative distribution function (CCDF) $F'(x)$. The mean is the average of all realizations of X and is sometimes called the expected value $E[X]$. The variance is a measure of how spread out different values of X might be. The standard deviation is simply the square root of the variance. The coefficient of variation is a normalized measure of dispersion of a probability distribution and is defined as the ratio of the standard deviation to the mean. The CDF shows the probability that a random variable X with a given probability distribution will be found at a value less than or equal to x.

The CCDF is opposite to the CDF and is used to answer the question how often the random variable X is above x. All of these marginal properties are calculated as

- mean $\mu = \frac{\sum_{i=1}^{n} X_i}{n}$,
- variance $\sigma^2 = \frac{\sum_{i=1}^{n} (X_i - \mu)^2}{n-1}$,
- standard deviation $\sigma = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \mu)^2}{n-1}}$,
- coefficient of variation $C_v = \frac{\sigma}{\mu}$,
- CDF $F(x) = P\{X \leq x\}$,
- CCDF $F'(x) = P\{X > x\} = 1 - F(x)$.

2.1.2 Autocorrelation

The autocorrelation function (ACF) of a stochastic point process $X = \{X_n\}$ describes the correlations between values of X at different points in time [48]. If X is second-order stationary, its ACF is defined as

$$R(k) = \frac{E[(X_i - \mu)(X_{i+k} - \mu)]}{\sigma^2}, \tag{2.1}$$

where $E[\cdot]$ is the expected value, k is the time shift being considered (usually referred to as the lag), and μ and σ^2 are the mean and the variance, respectively, of X.

2.1.3 Cross-correlation

In addition to studying how values of the same stochastic point process are correlated with each other via the ACF, it is also interesting to investigate the correlation between values of two distinct stochastic point processes $X = \{X_n\}$ and $Y = \{Y_n\}$. Such a correlation between X and Y is called the cross-correlation and can be shown via plotting sample values of X and Y jointly in a 2D-scatter plot. Figure 2.1 gives an example of showing the cross-correlation with a 2D-scatter plot, where the cross-correlation in Figure 2.1(a) is clearer and stronger than that in Figure 2.1(b) since most joint sample values of X and Y in Figure 2.1(a) are distributed over a straight line. It is also possible to quantitatively measure the cross-correlation between X and Y via calculating their correlation coefficient as

$$\rho_{X,Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}. \tag{2.2}$$

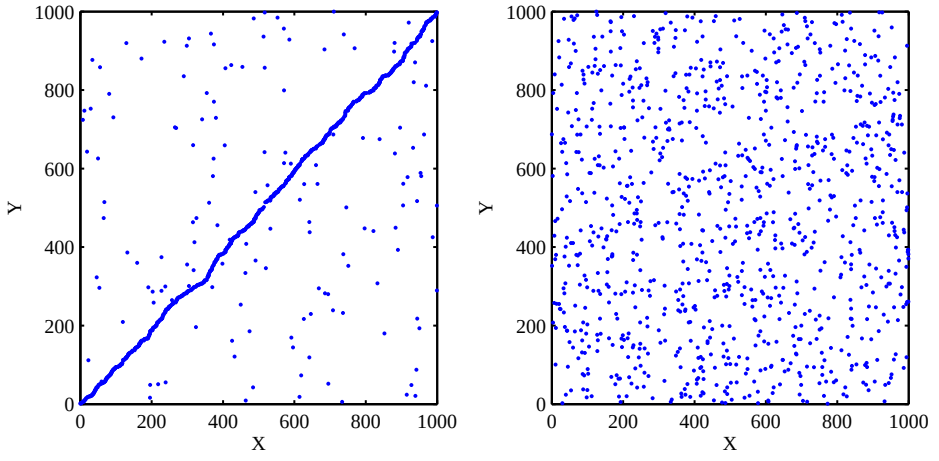


FIGURE 2.1: Example of the cross-correlation between two stochastic point processes, shown by their 2D-scatter plot.

The calculation of $\rho_{X,Y}$ in Eq. 2.2 is called the Pearson correlation coefficient. The cross-correlation between X and Y can also be checked by Spearman's approach. Spearman correlation coefficient is defined as the Pearson correlation coefficient applied to the ranks of data instead of the data itself. The values of each variable are sorted: the lowest value with rank 1, the next lowest value with rank 2 and so on. Equal values are given an averaged rank, for example, if there are two equal values that are ranked 5 and 6, they are both given rank 5.5.

2.2 Representations of Arrival Events

In a system workload, an arrival process, which refers to either arrivals of jobs or of Bags-of-Tasks, can be described as a series of individual time events $\{t_n\}$, where a job or a Bag-of-Tasks arrives on each time t_i. There exist different representations of an arrival process [48] as illustrated in Figure 2.2. Firstly, as an interarrival process which is a sequence $\{I_n\}$ with $I_n = t_n - t_{n-1}$. Secondly, as a count process which depends on a pre-selected time interval T and is obtained by dividing the time axis into equally spaced continuous intervals of T to yield a sequence of counts $\{C_n\}$, where C_n is the number of time events in the nth interval. Thirdly, as a rate process $\{R_n\}$ that is based on the count process, where $R_n = C_n/T$. We call each distinct value of T a time scale. With different time scales, we can represent the same arrival process by different count or rate processes. Any count or rate process with a time scale T can be converted to an arrival process by the *integrate-and-fire* algorithm [107].

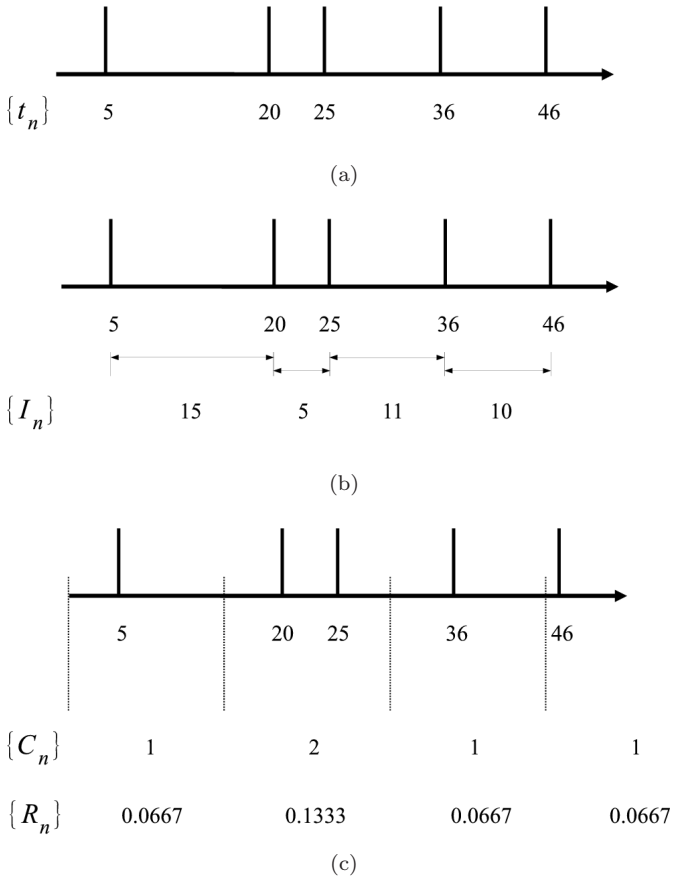


FIGURE 2.2: An illustration of representing an arrival process (a) by an interarrival process (b) and a count or rate process (c) with $T = 15$.

Each representation has its own advantage and drawback [47]. Representing a point process with $\{C_n\}$ or $\{R_n\}$ causes a loss of information about the times between arrival events within an interval T. On the contrary, $\{I_n\}$ keeps the whole information and so can be used to recreate the original point process accurately. However, the direct correspondence between its index number and the absolute time is lost. This shortcoming of $\{I_n\}$ is the advantage of $\{C_n\}$ and $\{R_n\}$ in contrast.

2.3 Definitions of Workload Features

A comprehensive workload that can be used in performance evaluation of parallel system scheduling should contain at least three attributes including arrival time, run-

time and parallelism, where each attribute is represented as a point process. Hence, the term "workload" used in this thesis is a combination of three point processes: the arrival time process $\{A_n\}$, the runtime process $\{R_n\}$ and the parallelism process $\{P_n\}$, where a job i in the workload is represented by the tripple A_i, R_i and P_i. The term "workload feature" refers to the characteristics of the point processes forming the workload, which are described in the rest of this section.

2.3.1 Features of an Arrival Process

Three characteristics of arrival events are studied, namely long range dependence, periodicity and temporal burstiness.

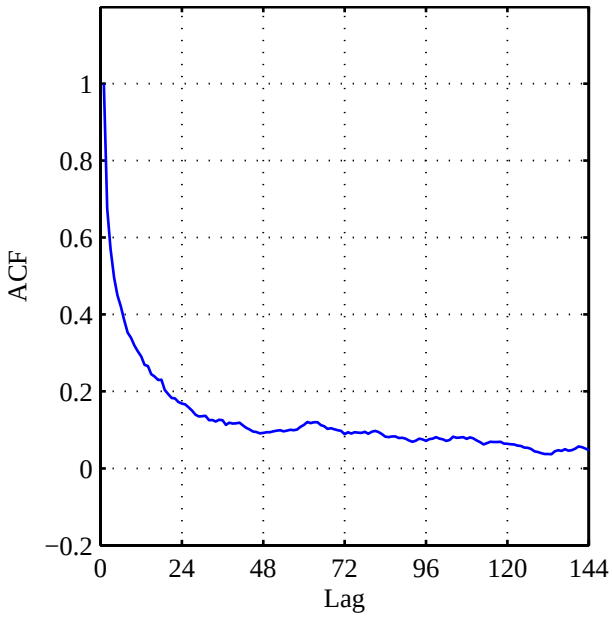


FIGURE 2.3: An illustration of long range dependence.

Long Range Dependence

A discrete-time second-order stationary process $X(t)$ is said to be long range dependent (LRD) if its autocorrelation function $R(k)$ satisfies the condition

$$R(k) \sim ck^{2H-2} \text{ as } k \to \infty, \tag{2.3}$$

where c is a constant, H is the Hurst parameter [36], and $R(k)$ decays so slowly that

$\sum_{k=0}^{\infty} R(k) = \infty$. Figure 2.3 gives an illustration of a LRD process shown by its ACF. LRD can be quantitatively measured by the Hurst parameter H which takes on values from 0.5 to 1. A value of 0.5 indicates the absence of LRD. The closer H is to 1, the greater the degree of LRD. For a deeper understanding, see [6, 19, 84].

Periodicity

In the context of parallel systems, the characteristic periodicity is observed because jobs are often submitted in cycles. The length of a cycle can be in the order of hours, days, months, years or any time length. The daily cycle is the most known and widely recognized cycle since it appears in all practical workloads [19, 116]. Parallel system workload cycles typically show a higher activity during the day and a lower activity during the night. However, details including cycle lengths may differ among different workloads.

Detecting the periodicity of a point process can be done via its autocorrelation function. An illustration of a process with periodicity is given in Figure 2.4. We can visually observe cycles in its ACF. The autocorrelation at lag 0 is 1 because the process is compared with itself, thus, the ACF results an exact match. Then when the lag increases, the autocorrelation quickly decays. As we can see, the ACF repeats every 24 lags since the process tends to match itself whenever the lag is an integral multiple of 24. For a deeper understanding, see [19].

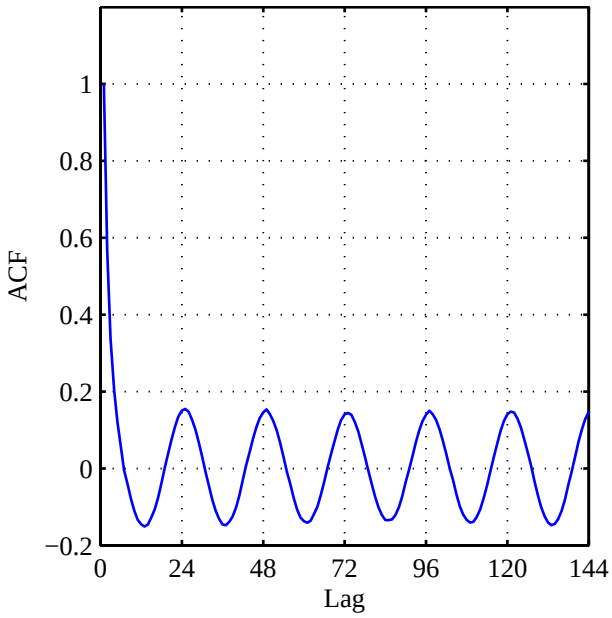


FIGURE 2.4: An illustration of periodicity.

Temporal Burstiness

In the context of system job traffic, we refer to temporal burstiness as the tendency of job arrivals to occur in bursts, separated by long periods of no arrivals. Since there exists no generally accepted definition for temporal burstiness despite its prevalence [42], we find that a definition that is closest to our work should be based on the concepts of burst and gap. A burst consists of job arrivals whose interarrival times are small while a gap refers to a large interarrival time. An illustration of our definition of temporal burstiness in job traffic is given in Figure 2.5, where we show two job arrival processes with different degrees of temporal burstiness. Job arrivals 2 are more bursty than job arrivals 1 because they have tighter bursts and longer gaps. For the concept of temporal burstiness in this thesis, we use the coefficient of variation C_v of interarrival times as a metric. If a job arrival process exhibits tight bursts and long gaps, it will result in a large C_v, indicating a large degree of temporal burstiness.

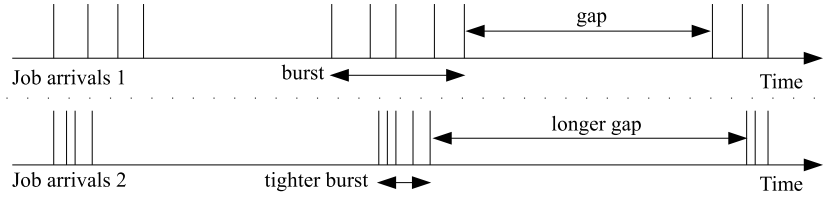


FIGURE 2.5: An illustration of temporal burstiness in two system job traffics.

2.3.2 Features of Runtime and Parallelism

Three characteristics are presented in this section, consisting of the temporal locality structure of a runtime process, spatial burstiness and the correlation between runtime and parallelism.

Temporal Locality

The phenomenon of temporal locality [12, 18] has been recognized and recently studied in modeling parallel system workloads [54, 67]. In this context, it is understood as a persistent similarity between runtimes of consecutive jobs. Real parallel workload data are far from being independently distributed, instead, jobs tend to arrive on parallel systems in bags referred as Bag-of-Tasks (BoT) behaviour. It is shown in Section 3.4 that a large number of jobs are submitted as part of BoTs in real parallel system workloads and jobs within the same BoT have similar runtimes. This means there are quite a lot of repetitions in a real runtime process. For example, consider the following 3 BoTs which are consecutively submitted to a system: the first BoT

consists of 4 jobs with runtimes $R_1 = \{10, 12, 15, 9\}$, the second BoT has $R_2 = \{3000, 2800\}$ and the third BoT contains $R_3 = \{400, 360, 420\}$. These BoTs form a runtime process $R = \{10, 12, 15, 9, 3000, 2800, 400, 360, 420\}$. Assume we have an efficient approach (as is introduced later in Section 3.3.1) to classify R in such a way that similar runtimes are grouped to the same cluster[1]. Then we will have a series of cluster labels corresponding to R: $L = \{A, A, A, A, C, C, B, B, B\}$. We use L to form a series of lengths of repetitions: $LR = \{4, 2, 3\}$ because the cluster labels A, C and B are repeated 4, 2 and 3 times, respectively. As such, we can see from L and LR that the common BoT behaviour in parallel workloads will lead to the phenomenon of repetitions in a runtime process. The term "temporal locality" refers to this phenomenon and the bigger the elements of LR, the larger the degree of temporal locality.

Spatial Burstiness

Spatial burstiness of a parallel workload refers to the non-uniformity of the distributions of runtimes and parallelisms. This non-uniformity can be observed via drawing a 3D-histogram of the runtimes and the parallelisms. Figure 2.6 gives an example of spatial burstiness. Observing two 3D-histograms in the figure, we see that Workload2 has a low degree of spatial burstiness because of its relatively flat 3D-histogram in Figure 2.6(b) while Workload1 in Figure 2.6(a) has a higher degree of spatial burstiness.

For a quantitative measure of spatial burstiness, we propose a new entropy-based approach to quantify it in a parallel workload. In information theory, entropy is common in measuring uniformity. The entropy of a random variable X is defined [90] as

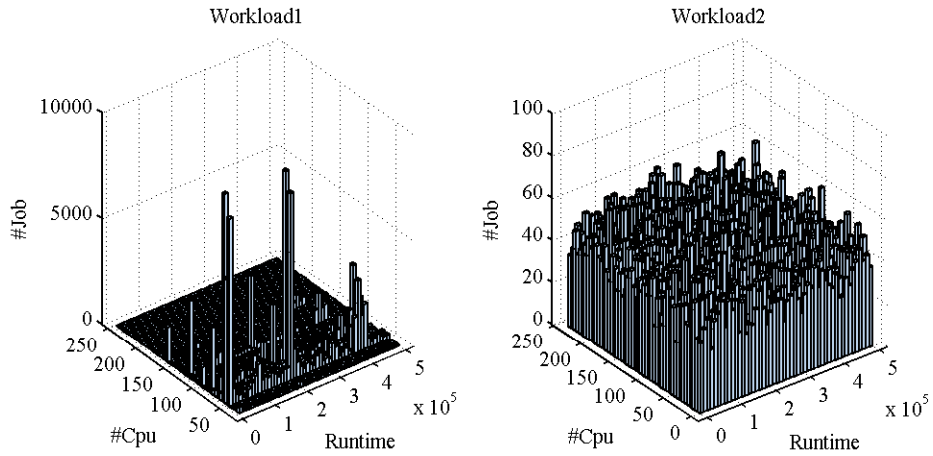
$$H(X) = -\sum_{i=1}^{N} p_i \times \log p_i, \tag{2.4}$$

where p_i indicates the probability for event X_i to happen. In our work, we measure spatial burstiness with a normalized entropy. It is known [33] that the entropy in Eq. (2.4) has a minimal value of 0 when for some j, $p_j = 1$ and $p_i = 0$, $i \neq j$. It reaches its maximal value of $\log N$ when $p_i = 1/N$, $i = 1, \ldots, N$. As such, $H(X)$ in general will increase with N. Hence, the normalized entropy H_{NE} of a random variable X, defined as

$$H_{NE}(X) = \frac{-\sum_{i=1}^{N} p_i \times \log p_i}{\log N}, \tag{2.5}$$

[1]The term "cluster" stems from the concept of "clustering". Clustering is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense [99].

will be bounded by 0 and 1. The disadvantage of quantifying spatial burstiness based on the normalized entropy is the dependency on the number of ranges N since it is necessary to divide the space axis into N ranges and calculate the probability for an event to happen on each range. If N changes, the probabilities will also change. This leads to different values of the normalized entropy and thus gives an instable measure[2]. However, with parallel workloads, we find an efficient method to eliminate this dependency by defining p_i in Eq. (2.5) flexibly. For a workload W, we calculate p_i as $p_i = TR_i/TR$, where TR_i is the total runtime of all jobs in W that request i processors and TR is the total runtime of all jobs in W. As such, the value of N is equal to the maximal number of processors that a job may request in W, and therefore the measure is stable. Furthermore, since the entropy is normalized, this metric only ranges from 0 to 1. The closer the metric is to 0, the stronger the spatial burstiness.



(a) (b)

FIGURE 2.6: Example of spatial burstiness of two workloads by using 3D-histograms to show the distribution of jobs according to their runtimes and parallelisms.

Correlation between Runtime and Parallelism

The correlation between runtime and parallelism can be checked simply by calculating their correlation coefficient. The value of a correlation coefficient ranges from -1 to

[2]Wang et al. in [110] used the entropy function in Eq. (2.4) to measure the temporal-spatial burstiness in I/O traffic. They use entropy plots to show that the entropies of disk traces nearly fit to a line when changing the number of ranges and they use the slope of the line as the metric for temporal-spatial burstiness. We tried their idea and failed because we found that the entropies of parallel workloads do not fit to a line.

+1. A negative value indicates that smaller runtimes tend to be associated with larger numbers of processors. This means large applications often use more processors to reduce their runtimes based on the implicit assumption that their total amount of work remains the same. Vice versa, a positive value shows the tendency of the association between smaller runtimes and smaller numbers of processors. If a workload exhibits a positive correlation, the implicit assumption above is no longer correct. It means that large applications often run longer with more processors.

As presented in Section 2.1.3, the correlation coefficient between two point processes can be calculated by using Pearson's or Spearman's approach. However, Pearson's approach is not strong enough to capture correlation since it works well under many assumptions and one is that the two processes follow normal distributions, but this is not the case for real world data such as job runtimes and numbers of processors. Spearman's approach, which uses the data ranks instead of the data itself to calculate the correlation coefficient, is more suitable to examine correlation, in particular when we want to know if smaller runtimes are associated with smaller or larger numbers of processors. Therefore, Spearman's approach is used in this research.

2.3.3 The Bag-of-Tasks Behaviour

We consider a parallel workload W as an ordered set of N jobs: $W = \{J_i | i = 1, \ldots, N$ and $AT(J_i) \leq AT(J_j)$ if $i < j\}$, where $AT(\cdot)$ denotes the arrival time. Since there is no general definition of the Bag-of-Tasks (BoT) behaviour for parallel workloads, we define a BoT with a time parameter Δ as a maximal contiguous subsequence B of W such that

1. For any two successive jobs J_i, J_{i+1} in B, we have $AT(J_{i+1}) \leq AT(J_i) + \Delta$.
2. All jobs in B have the same values with respect to user name, group name, queue name, job name, estimated runtime and number of processors.

Similar to a job, a BoT also has its own attributes which are defined as follows:

1. *The arrival time* of a BoT is the minimal arrival time of jobs within the BoT.
2. *The submission duration* of a BoT is the difference between the maximal and the minimal arrival times of jobs within the BoT.
3. *The size* of a BoT is the number of jobs in the BoT.
4. According to the BoT definition, except for the arrival time, all jobs in a BoT have exactly the same values for other attributes. Moreover, we indicate in Section 3.4 that they also have similar runtimes. Therefore, we argue that a BoT can be represented by any job that it contains. Hence, we define *the runtime*,

the parallelism and *the estimate* of a BoT as the average of the runtimes of all jobs, the requested number of processors and the user estimated runtime of any job in the BoT, respectively. We have chosen this definition because it can help to simplify modeling research. For example, a model can easily create all jobs in a BoT after determining the size, the runtime, the parallelism and the estimate of the BoT because we know the number of jobs in the BoT, the runtimes, the requested numbers of processors and the user estimated runtimes of the jobs via the size, the runtime, the parallelism and the estimate of the BoT. The remainder that the model needs to handle is to decide the interarrival times of the jobs.

2.4 Summary

This chapter provided a brief background knowledge for readers to follow the rest of the thesis. The background includes statistical theories that are used intensively in Chapter 3, where workload characterisation plays a central role. In addition, different representations of arrival events were introduced for use in investigation of job/BoT arrivals. Finally, several workload features were described; namely long range dependence, periodicity and temporal burstiness of job arrivals; temporal locality, spatial burstiness and correlation of runtimes and parallelisms; and the Bag-of-Tasks behaviour. The concepts of these features will be used broadly in workload modeling (Chapters 4 and 5) and performance evaluation (Chapters 6 and 7) studies.

Chapter 3

Statistical Analysis

Any research of modeling and performance evaluation that centers around workloads should begin with a study of workload characterisation. Therefore, this chapter focuses on a statistical analysis of parallel system workloads that will help to enrich the understandings of a system and the workload running on it. In addition, the analysis is also applied to grid traces to show that some features of parallel workloads can be observed on grid data and, thus, it is possible to apply research results in later chapters to grids. In particular, we emphasize the Bag-of-Tasks (BoT) behaviour because it has become more common in scheduling and modeling research [2, 39, 71, 109, 112]. According to the BoT definition, a BoT can consist of only one job. In our analysis, we only take into account BoTs that contain at least 2 jobs (so-called real BoTs), except for characterising BoT arrivals. BoTs with a single job (so-called unreal BoTs) are not excluded in this case because in practice, real and unreal BoTs arrive in a mixed way. Hence, excluding unreal BoTs in an arrival process causes a loss of information about interarrival times since arrival events of unreal BoTs are discarded.

The chapter firstly introduces the data used in the analysis and provides information about where the data are collected and how they are pre-processed so that research results presented in this thesis can be reproduced. After that, the workload features introduced in Chapter 2 are analyzed statistically. Finally, we discuss their important roles in clusters, grids and clouds to indicate that it is essential to take them into modeling and evaluate their performance impacts on parallel systems.

3.1 Workload Data under Study

Our study covers more than 2.5 million real grid and parallel jobs. These include ~1.3 million jobs from fourteen parallel traces. For the traces, we only take into account jobs that finished successfully because the runtimes of jobs that have not finished are not known. Summary details of the traces are given in Table 3.1, where we see that most parallel systems are scheduled by Maui [61], besides Slurm [91], Catalina [7] and LSF [58]. In addition to parallel workloads, two grid traces are included to

illustrate that part of our work can be applied successfully for grid jobs. Note that we remove from NorduGrid the first 3 jobs due to the abnormal long interarrival time (∼5 months) between the third and fourth jobs. The parallel and grid traces, except for NIKHEF, can be obtained from the Parallel Workloads Archive [80] and the Grid Workloads Archive [31], respectively. Note that, there are two versions for every logged trace on the Parallel Workloads Archive. The original version is the collected data while the cleaned version removes a number of jobs from the original version because the original data often includes problematic and unrepresentative data, such as significant automated administrative activity or large-scale flurries of activity by single users [80]. In our study, we use the cleaned version for all parallel workloads as recommended on the Parallel Workloads Archive, which is based on [23, 106]. The cluster NIKHEF is located at the High Energy Physics institute in the Netherlands, which participates in the LCG grid [44]. The names of the other traces are equal to those mentioned on the two websites [31, 80] so that their full details can be retrieved easily.

Table 3.1: Summary of cluster and grid data used in experimental study.

Trace (Abbreviation)	Period	CPUs	Jobs	Scheduler	Type
NorduGrid (NOR)	09/03-04/06	2000	781367	Grid Broker	Grid
AuverGrid (AUV)	12/05-12/06	475	404176	Grid Broker	Grid
DAS2 fs0 (FS0)	01/03-01/04	144	192269	Maui	Cluster
DAS2 fs1 (FS1)	01/03-12/03	64	34808	Maui	Cluster
DAS2 fs2 (FS2)	01/03-12/03	64	61040	Maui	Cluster
DAS2 fs3 (FS3)	01/03-12/03	64	62767	Maui	Cluster
DAS2 fs4 (FS4)	02/03-12/03	64	29563	Maui	Cluster
OSC Cluster (OSC)	01/00-11/01	57	36096	Maui	Cluster
HPC2N (HPC)	07/02-01/06	240	201998	Maui	Cluster
NIKHEF (NIK)	01/04-04/05	288	220575	Maui	Cluster
LLNL uBGL (LLU)	11/06-06/07	2048	11280	Slurm	Cluster
LLNL Atlas (LLA)	11/06-06/07	9216	21179	Slurm	Cluster
LLNL Thunder (LLN)	01/07-06/07	4096	102972	Slurm	Cluster
SDSC BLUE (SDB)	04/00-01/03	1152	195591	Catalina	Cluster
SDSC DataStar (SDD)	03/04-04/05	1664	66743	Catalina	Cluster
LANL O2K (LAN)	11/99-04/00	2048	93326	LSF	Cluster

3.2 Job Arrival Analysis

We draw in Figure 3.1 different statistics of job arrivals from a grid trace and a cluster trace. As we can observe from the autocorrelation functions (ACFs), if represented as an interarrival process, job arrivals do not exhibit a correlation structure. However, if represented by a rate process, job arrivals do exhibit the long range dependence (LRD) feature clearly. This is consistent with the study in [46], which concludes that LRD of system job arrivals should be reliably revealed in count/rate-based measures. This is the reason why we use the rate process representation for LRD of job arrivals.

We quantitatively measure the degree of LRD in the real traces by estimating the Hurst parameter and show the results in Table 3.2. We select three estimators, namely *Aggregate Variance, R/S Statistic* and *Periodogram* [101], and calculate the mean and the standard deviation of their estimates. We can see from Table 3.2 that LRD is present strongly in real job arrivals since the estimated results are much larger than 0.5.

TABLE 3.2: The Hurst parameter of rate processes (T=900 seconds).

Trace	Hurst parameter
NOR	0.86 ± 0.07
AUV	0.91 ± 0.06
FS2	0.77 ± 0.10
FS3	0.84 ± 0.09
HPC	0.70 ± 0.09
LAN	0.80 ± 0.03
LLN	0.69 ± 0.17
NIK	0.70 ± 0.08

With respect to temporal burstiness, we argue that it has to be exhibited in real job arrivals due to the occurrence of Bags-of-Tasks and idle periods during nights, weekends, holidays, etc. when users often submit fewer jobs. As we can see from Figure 3.1, the interarrival time distributions of both grid and cluster traces are heavy-tailed and so the temporal burstiness is observed. We measure the temporal burstiness degree in real job arrivals in Table 3.3 by applying the metric coefficient of variation C_v to interarrival times. As a distribution with a C_v smaller than 1 is considered to have low variance, the results confirm a common presence of temporal burstiness in system job traffic since they are much larger than 1.

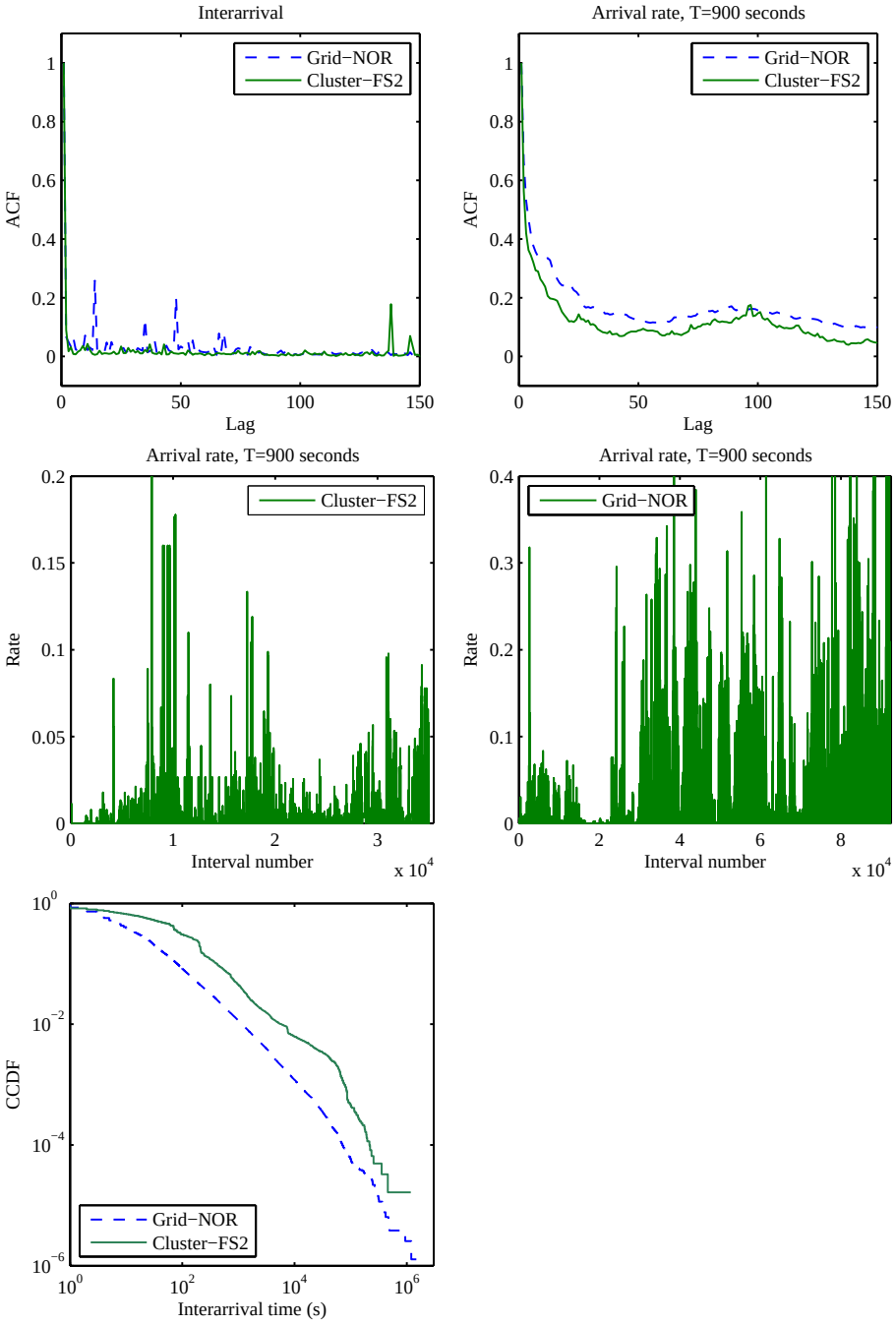


FIGURE 3.1: Statistics of grid (NOR) and cluster (FS2) job arrivals under two different representations.

TABLE 3.3: Temporal burstiness of cluster/grid job arrivals, expressed by the coefficient of variation.

Trace	Coefficient of variation
NOR	33.14
AUV	5.65
FS2	13.88
FS3	17.92
HPC	5.70
LAN	17.70
LLN	10.00
NIK	18.56

3.3 Runtime and Parallelism

This section shows a common presence of the correlation between runtime and parallelism as well as the features temporal locality and spatial burstiness in a large number of real traces. According to the definition of temporal locality, we need an efficient approach to classify a job runtime process in such a way that similar runtimes are grouped to the same cluster. Therefore, we start this section by introducing such a classification framework and continue by examining the three characteristics in real workloads.

3.3.1 Runtime Classification

In our study, runtime classification is done by using Model-Based Clustering (MBC) [28]. It is shown in [54] that MBC is an efficient choice to classify a runtime process and obtain a series of cluster labels. MBC is a methodological framework that underlies a powerful approach not just to data clustering but also to discriminant analysis and multivariate density estimation. Instead of looking for a single probability density function for the data distribution, the main idea of MBC is that it considers the data as generated by a mixture of normal (Gaussian) probability density functions, where each function represents a different cluster. The selection of the number of clusters is based on the Bayesian information criterion [28]. Gaussian parameters for these clusters are calculated by combining agglomerative hierarchical clustering and the expectation-maximization algorithm for maximum likelihood [28]. MBC is implemented in the MCLUST software and available on [62].

3.3.2 Examination of Correlation

We calculate the correlation between runtime and number of processors in real workloads and show the results in Table 3.4. As one can see, the correlation feature commonly exists in real world data because 10 of 12 traces exhibit correlations with 4 negative and 6 positive, while only 2 traces show that runtime and parallelism are not correlated with correlations around 0. A notable point is that negative correlations concentrate around the value of -0.2 and for positive correlations around $+0.4$. Therefore, when we study the performance issues of locality, we will focus on three realistic degrees of correlation, namely -0.2, 0 and $+0.4$, as is shown later in Chapter 7.

TABLE 3.4: The feature correlation between runtime and parallelism of real parallel workloads, expressed by applying Spearman's approach.

Trace	Spearman's Correlation
FS3	-0.247
HPC	-0.229
FS4	-0.214
OSC	-0.202
FS0	0.002
SDD	0.005
FS1	$+0.269$
SDB	$+0.409$
LLA	$+0.431$
LLU	$+0.444$
LLN	$+0.453$
FS2	$+0.459$

3.3.3 Examination of Temporal Locality

As described in Section 2.3.2, the phenomenon of temporal locality is mainly caused by the Bag-of-Tasks behaviour which leads to the repetitions of similar runtimes in a runtime process. Feitelson [18] fitted the lengths of the repetitions with a Zipf distribution, defined by its probability density function

$$P(x) \sim \frac{1}{x^{\alpha}} \text{ with } \alpha > 0, \tag{3.1}$$

where α is a shape parameter. As from Eq. (3.1), the larger the value of α, the exponentially smaller the probability for the occurence of a big length of repetitions and thus the smaller the degree of temporal locality.

TABLE 3.5: The feature temporal locality of real parallel workloads, expressed by estimating the shape parameter α of a Zipf distribution.

Trace	Shape parameter α
FS3	1.88
HPC	2.26
FS4	1.74
OSC	2.48
FS0	2.45
SDD	2.91
FS1	2.07
SDB	3.17
LLA	2.62
LLU	2.71
LLN	2.92
FS2	1.98

By applying the Model-Based Clustering framework on real traces, we obtain series of cluster labels and based on that, we calculate series of lengths of repetitions. We present the temporal locality feature of real traces in Figure 3.2 by drawing the log-log histograms of lengths of repetitions. Visually, the histogram of a Zipf distribution will show a straight line with a negative slope using log-log axes. However, the tail of the Zipf distribution is hard to characterise because there are many big sizes that each appears only a few times, and thus it shows more diversity at the tail. As observed from Figure 3.2, the lengths of repetitions in the real world data are fitted well to Zipf distributions and thus the temporal locality feature of the real traces can be determined by estimating the shape parameter α of the Zipf distributions[1].

[1]Note, we only use the shape parameter α of a Zipf distribution to determine the phenomenon of temporal locality and to estimate how large lengths of repetitions of a runtime process are. We do not recommend α as a quantitative metric of locality. We refer to [18] for such a metric.

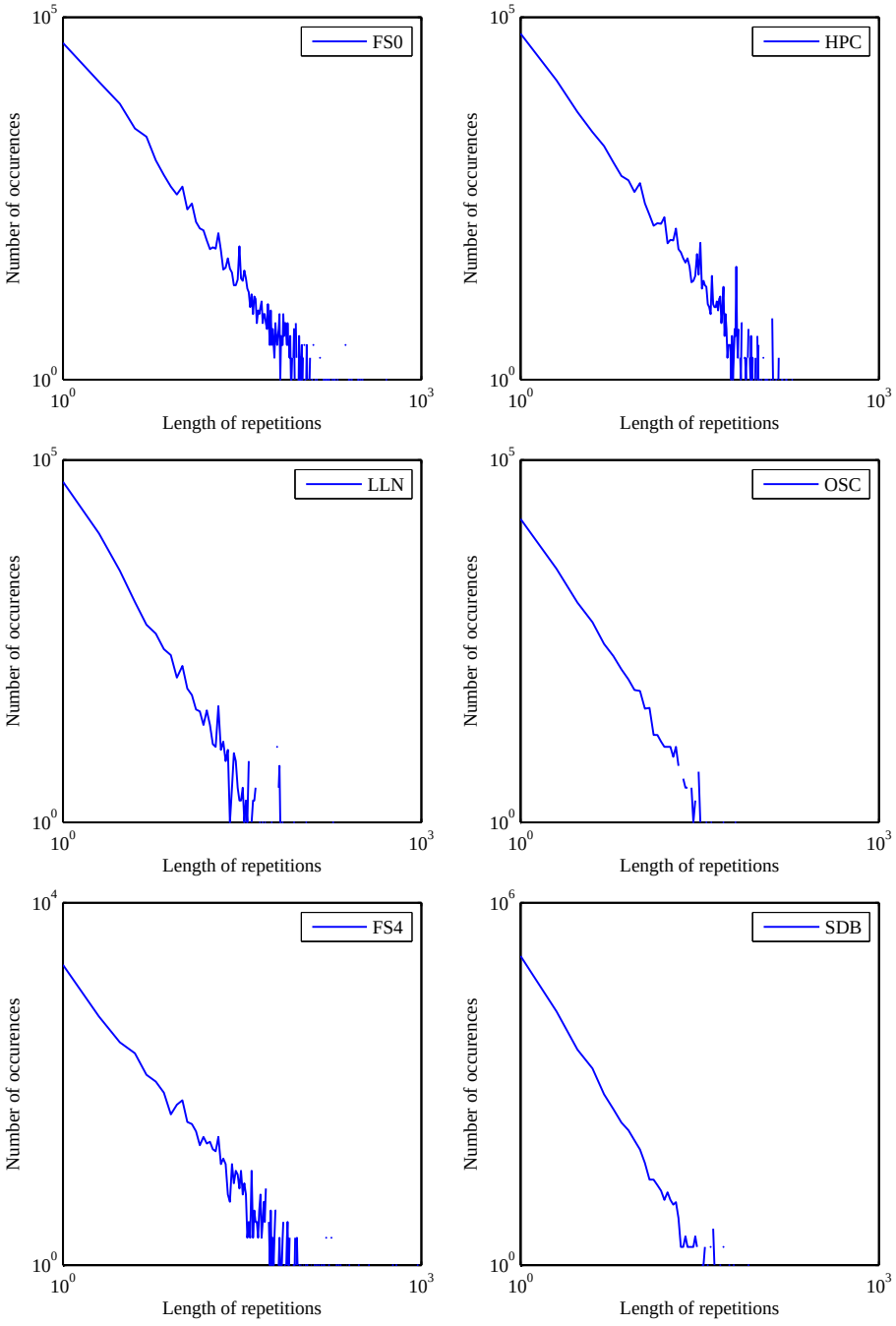


FIGURE 3.2: Log-log histograms of lengths of repetitions of similar runtimes in six real traces.

The estimated results of α are shown in Table 3.5. It can be seen that the temporal locality property exists commonly in real traces and its degree varies among the real workloads. Therefore, we believe that temporal locality deserves to be taken into account to evaluate its effect on scheduling.

3.3.4 Examination of Spatial Burstiness

To characterise spatial burstiness in real parallel workloads, we apply the approach of using normalized entropy proposed in Section 2.3.2 and show the results in Table 3.6. An interesting observation from the results is that the feature spatial burstiness exists clearly in real data because the results are closer to 0 than 1. This observation suggests that this property deserves to receive more attention in the literature. The study on the impact of this characteristic on scheduling should be emphasized so that correct workloads are used in evaluating scheduling algorithms.

TABLE 3.6: The normalized entropy of spatial burstiness.

FS2	FS3	HPC	LAN	LLN
0.447	0.126	0.305	0.317	0.330

3.4 Basic Analysis of Bag-of-Tasks

This section analyzes many basic features of the Bag-of-Tasks behaviour. As noticed in the BoT definition, forming a BoT depends on the parameter Δ. Therefore, in order to determine a suitable value for this parameter, we define the set S of a workload W as including all interarrival times (IATs) between any two successive jobs J_i, J_{i+1} in W such that J_i and J_{i+1} have the same values for user name, group name, queue name, job name, estimated runtime and number of processors. According to the BoT definition, J_i and J_{i+1} can belong to the same BoT. However, if J_i terminates before J_{i+1} arrives, we exclude their IAT from S because when users know the result of J_i, they may adjust J_{i+1} and therefore, J_i and J_{i+1} should not belong to the same BoT. In Figure 3.3, we draw the cumulative distribution functions (CDFs) of all IATs for all six parallel traces. From the figure we see that most IATs do not exceed 100 seconds. Therefore, we select $\Delta = 100$ seconds in our study because larger values do not much increase a BoT size but reduce the meaning of a BoT.

The next question regarding BoTs is whether they are common enough to be taken into account in modeling. We thus calculate the number of jobs submitted as part of BoTs in parallel traces. Applying $\Delta = 100$, we find that BoT submissions are common in parallel systems because on average in the six traces, 62% of jobs are submitted

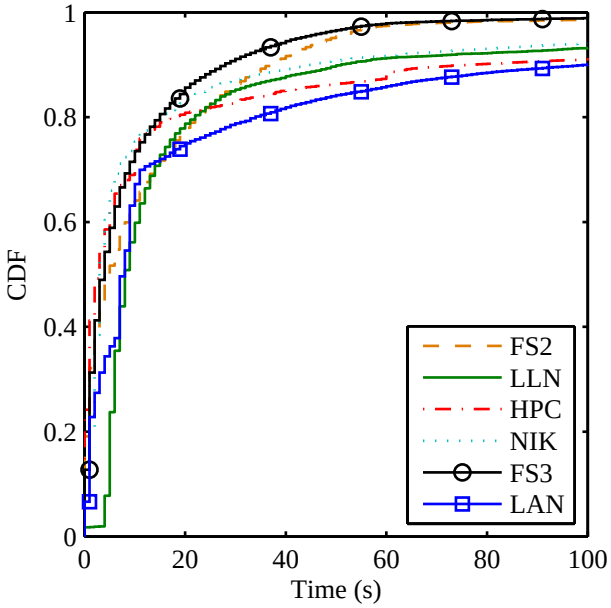


FIGURE 3.3: The cumulative distribution functions of IATs in six parallel traces.

TABLE 3.7: Fraction of jobs submitted as part of BoTs.

FS2	FS3	HPC	LAN	LLN	NIK
60%	89%	65%	34%	40%	83%

as part of BoTs, as is shown in Table 3.7. Hence, we conclude that BoTs should be incorporated into a workload model.

Another issue of BoTs that could be important for modeling is the runtimes of jobs within a BoT. We have reason to believe that the runtimes will be similar because the jobs have the same values for other attributes such as user and group names, etc. To check this, we compute the average coefficient of variation C_v of the runtimes in the BoTs. As a distribution with a $C_v < 1$ is considered to have low variance, runtimes in a BoT exhibit low variance since the average C_v is smaller than 1 and closer to 0 as shown in Table 3.8. Therefore, we conclude that jobs within a BoT have similar runtimes.

TABLE 3.8: The average coefficient of variation of the runtimes of the jobs in a BoT.

FS2	FS3	HPC	LAN	LLN	NIK
0.19	0.20	0.25	0.28	0.45	0.46

The question of the length of the submission duration of a BoT is also interesting. We define the submission duration of a BoT as the difference between the maximal and the minimal arrival times of jobs within the BoT. To determine how big in time a BoT is, we draw the cumulative distribution functions of these durations of the real data. As we can see from Figure 3.4, for all three workloads, almost 100% of the BoT submission durations are smaller than 15 minutes.

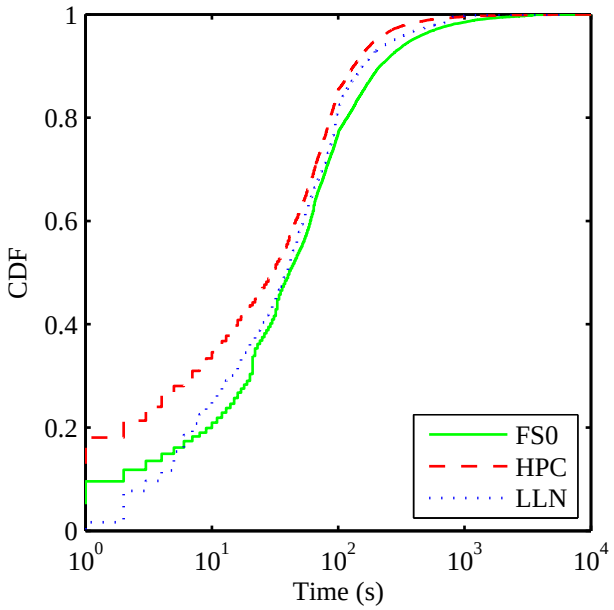


FIGURE 3.4: The cumulative distribution functions of BoT submission durations in the real workloads.

The last BoT-related issue that could be taken into account in a workload model is the distribution of BoT sizes. As we indicate in Section 3.3.3, runtimes of parallel workloads tend to be similarly repetitive and Feitelson [18] shows that runlengths[2] can be fitted by a Zipf distribution. Since job runtimes of a BoT are similar, it is possible that a Zipf distribution can also be applied for BoT sizes. This is indeed confirmed in Figure 3.5.

3.5 BoT Arrival Analysis

We introduce in Section 2.2 that an arrival process can be represented by either an interarrival process or a count/rate process. In this section, we start with the

[2]He calls the length of such a repetition of similar runtimes a runlength.

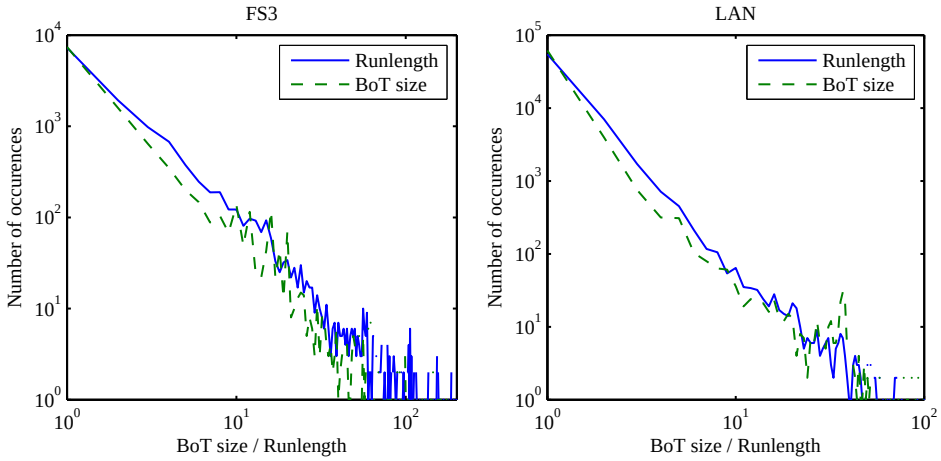


FIGURE 3.5: Log-log histograms of BoT sizes (including sizes equal to 1) and runlengths in real traces.

analysis of BoT arrivals in terms of their interarrival times to characterise the temporal burstiness. Then, we represent BoT arrivals by a rate process to characterise long range dependence and periodicity features.

3.5.1 Interarrival Times and Temporal Burstiness

The cumulative distribution functions of interarrival times of six traces are drawn in Figure 3.6(a). For each trace, the BoT interarrival times are fitted to the following five distributions which are used extensively in workload analysis [19]: Generalized Pareto (GP), Weibull (Wbl), Lognormal (LogN), Gamma (Gam) and Exponential (Exp). The Maximum Likelihood Estimation (MLE) method [76] is used to estimate the parameters for those distributions in the fitting process, which is done with a confidence level of 95%. Table 3.9 shows the results of this fitting process. For each distribution with the estimated parameters in Table 3.9, we use a Goodness-of-Fit test, called Kolmogorov-Smirnov (KS test) [19], to assess the quality of the fitting process. Based on the KS statistic D produced by the KS test, we will determine the best distribution that fits to the real data. D estimates the maximal distance between the CDF of the fitted distribution and that of the empirical data. A smaller value of D (closer to 0) indicates a better fit between the empirical data and the tested distribution. Iosup et al. [39] found out that the Weibull distribution is the best fit for interarrival times of BoTs in grid workloads. However in the context of parallel system workloads, our study shown in Table 3.10 indicates that although the Weibull distribution is a good candidate, it is not the best fit. Instead, the Generalized Pareto distribution seems to be the best choice since its KS statistic is the smallest in five over six cases. For HPC, though KS statistic of the Generalized Pareto distribution

is larger than that of the Weibull distribution, KS statistics of both distributions are close to 0 and therefore both distributions can be fitted well to the data as shown in Figure 3.6(b). From this result, we argue that the Generalized Pareto distribution should be used for modeling BoT arrivals.

TABLE 3.9: Parameters of distributions estimated during the fitting process. $a, b, \theta, \mu, \sigma$ indicate shape, scale, threshold, mean and standard deviation, respectively.

Dist.	GP(a,b,θ)	Wbl(a,b)	LogN(μ,σ)	Gam(a,b)	Exp(μ)
FS0	1.12 67.53 0	0.51 170.36	4.02 3.13	0.35 1100	383.71
FS2	0.81 165.99 0	0.38 252.31	3.73 5.52	0.22 4887	1085.29
FS3	1.26 132.94 0	0.39 374.64	4.48 4.09	0.21 12826	2723.14
HPC	1.26 172.2 0	0.47 475.43	4.91 3.46	0.31 3874	1213.38
LLN	0.96 40.23 0	0.57 91.98	3.6 2.32	0.41 438	180.02
NIK	1.22 109.27 0	0.29 166.28	2.47 7.23	0.19 3101	589.2

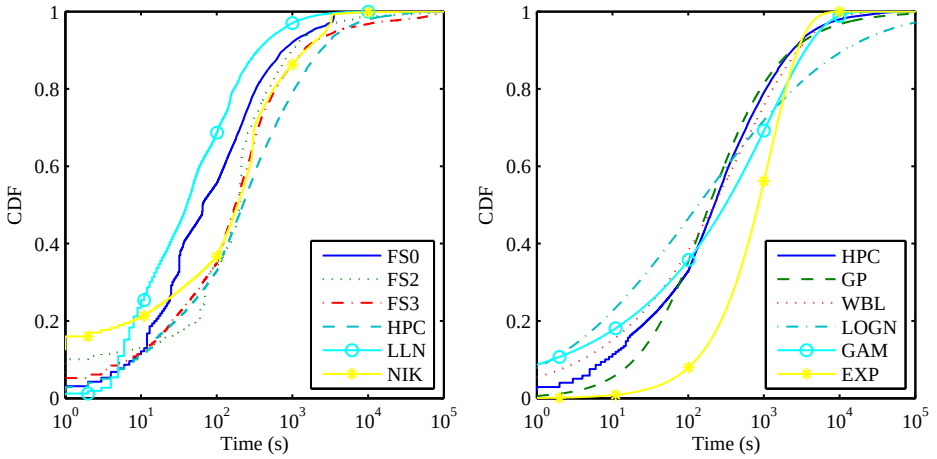
TABLE 3.10: KS statistics obtained from KS tests.

Dist.	GP	Wbl	LogN	Gam	Exp
FS0	0.059	0.098	0.182	0.118	0.348
FS2	0.098	0.237	0.323	0.212	0.478
FS3	0.063	0.106	0.192	0.237	0.604
HPC	0.074	0.054	0.139	0.103	0.355
LLN	0.063	0.127	0.143	0.137	0.322
NIK	0.152	0.212	0.284	0.197	0.286

A good fit of the Generalized Pareto distribution to BoT interarrival times in real data implies that BoT arrivals are bursty. We quantitatively measure the temporal burstiness feature using the metric coefficient of variation C_v. This metric is equal to 0 if arrivals are not bursty. The larger this metric is, the burstier the arrivals are. The results in Table 3.11 show that BoTs are submitted to parallel systems in a bursty way.

3.5.2 Long Range Dependence and Periodicity

The periodicity and the long range dependence of an arrival process, represented by a rate process, can be observed via the autocorrelation function of the rate process. In



(a) (b)

FIGURE 3.6: Cumulative distribution functions of BoT interarrival times of real workloads (a) and of HPC2N and fitted distributions (b).

TABLE 3.11: Temporal burstiness of BoT arrivals expressed as the metric coefficient of variation.

FS0	FS2	FS3	HPC	LLN	NIK
7.7	9.6	7.7	3.8	8.3	9.9

our study, we characterise both BoT and job arrivals to achieve a comparison of how different they arrive on parallel systems. We select a time scale equal to 15 minutes for converting an arrival process to a rate process because we have shown in Section 3.4 that almost 100% of the BoT submission durations are smaller than 15 minutes.

We can visually observe the periodicity and the LRD properties in Figure 3.7. Since we select a time scale of 15 minutes, daily or weekly cycles of a rate process can be detected if its autocorrelation function repeats every 96 or 672 lags, respectively. Moreover, we also quantitatively measure the LRD using the Hurst parameter as metric and show the results in Table 3.12. Note that the degree of LRD exponentially increases when H is closer to 1. As we can see from Figure 3.7 and Table 3.12, several patterns of arrivals are identified. For NIK, both BoT and job arrivals do not have periodicity and exhibit the same weak dependence. In contrast for LLN, both BoT and job arrivals have clear periodicity with a weekly cycle. Furthermore, they also show the same large degree of LRD. For HPC, job arrivals have weak dependence and no periodicty while BoT arrivals exhibit larger dependence and a clear daily cycle. For FS3, both BoT and job arrivals have the same degree of LRD but the daily cycle is only observed in case of BoTs. For FS2, BoT arrivals have a large dependence

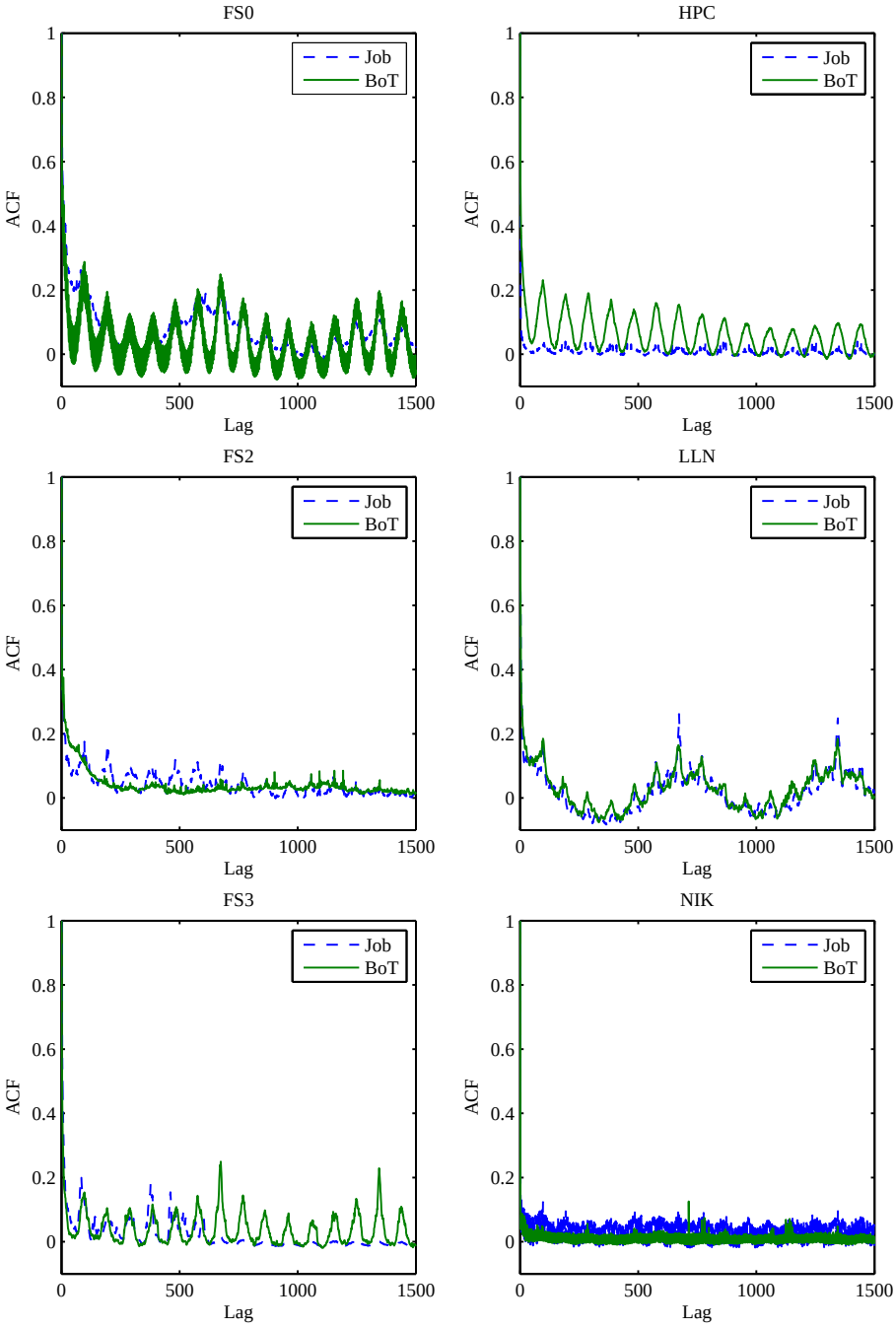


FIGURE 3.7: Autocorrelation functions of BoT and job arrivals.

but no periodicity while job arrivals have a smaller degree of LRD and exhibit a weak daily cycle. Finally for FS0, job arrivals show a larger dependence and a longer cycle (weekly versus daily) comparing with BoT arrivals. As such, it is clear that job traffic in parallel systems is a complex problem. BoT arrivals can have similar structures as job arrivals, but they can also be completely contrary. Therefore, we believe that traffic models in parallel systems should take care of this problem and more research on modeling job traffic should be done to provide models that are as realistic as possible.

TABLE 3.12: LRD of job and BoT arrivals expressed by estimating the Hurst parameter H.

	FS0	FS2	FS3	HPC	LLN	NIK
Job	0.85	0.79	0.74	0.67	0.78	0.69
BoT	0.76	0.86	0.73	0.81	0.79	0.68

3.6 BoT Size, Runtime, Parallelism and Estimate

In this section, we focus our analysis on four attributes of BoTs, namely size, runtime, parallelism and estimate. The characterisation will concentrate on examining the autocorrelations and the cross-correlations between these BoT attributes. Figure 3.8 shows that BoT runtimes can exhibit weak to strong autocorrelations. The autocorrelation in the sequence of BoT runtimes occurs because users tend to submit the same applications over and over again. This behaviour of users also causes the autocorrelation of job runtimes. However, calculating the autocorrelation in the sequence of job runtimes will be affected by the repetitions of similar jobs in the same BoT. Consequently, it yields larger autocorrelations and job runtimes become more sensitive with the autocorrelation function at short lags. Therefore, we argue that the autocorrelation should be calculated based on BoT runtimes instead of job runtimes.

With respect to the attribute BoT size, we investigate and show in Table 3.13 its statistics consisting of the mean and the maximum size. A noticeable point is that BoT sizes of 4 (FS0, FS2, HPC and NIK) out of 6 traces have similar means. Furthermore, the maximum size of a BoT is rather large and can be up to thousands of jobs. Parallel systems can undergo durations of severe congestion when such a large BoT occurs. Hence, we claim that realistic BoT workloads used in scheduling evaluation should contain large BoTs with hundreds to thousands of jobs for a reliable evaluation result.

It is also interesting to see how BoT sizes are correlated with other attributes of BoTs by calculating the Spearman correlation coefficient. Firstly, we examine the correlation between the size and the runtime. Before doing the examination, we

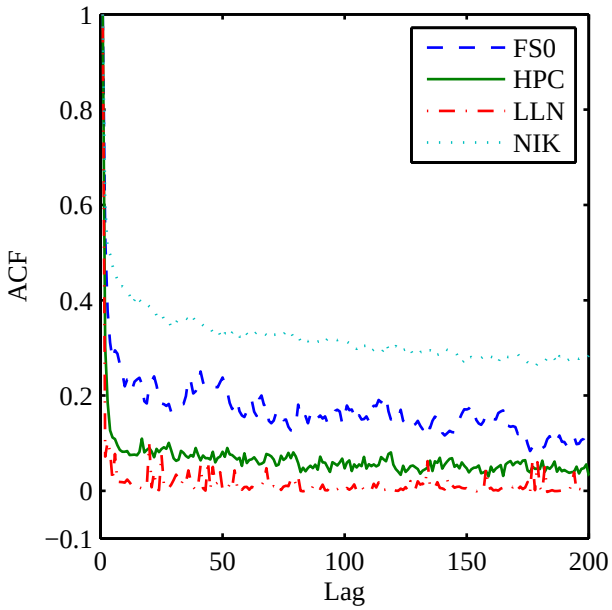


FIGURE 3.8: Autocorrelation functions of BoT runtimes.

TABLE 3.13: Statistics of BoT sizes.

	FS0	FS2	FS3	HPC	LLN	NIK
Mean	7	8	14	7	5	8
Max	1263	558	3675	2418	169	1201

expected that BoT runtimes would decrease if BoT sizes increase. This should give negative correlations, because we think that users would divide their applications into several smaller jobs by increasing their BoTs. However, results in Table 3.14 tell us that our expectation seems to be correct only for NIK. For FS2 and LLN, the correlations are also negative but rather weak, and in contrast FS0, FS3 and HPC show positive correlations. This means that if users increase their BoT sizes, jobs tend to run longer and this will harm the performance of parallel systems. Therefore, we believe that modeling and scheduling studies should take care for this realistic situation.

TABLE 3.14: Correlation between BoT sizes and BoT runtimes.

FS0	FS2	FS3	HPC	LLN	NIK
0.169	-0.038	0.201	0.157	-0.082	-0.106

Our next expectation is that BoT parallelisms will decrease if BoT sizes increase. Results of calculating the correlation between the two attributes, shown in Table 3.15, confirm our expectation in case of FS2, FS3 and HPC since their correlations are negative (the other traces also produce negative correlations but rather weak). We predict this result because we believe that users should reduce the numbers of requested processors when they increase their BoT sizes. Otherwise, there could be not enough free processors to be allocated to their jobs.

TABLE 3.15: Correlation between BoT sizes and BoT parallelisms.

FS0	FS2	FS3	HPC	LLN	NIK
-0.032	-0.198	-0.310	-0.150	-0.091	-0.014

Finally, we calculate the correlation between the size and the estimate and show results in Table 3.16[3]. As we see, the correlation is negative in case of FS2, FS3 and NIK. This means that users of these systems tend to take initiative to reduce the amount of time they request for their jobs when they increase the size of BoTs, possibly because they do not want schedulers to let their jobs wait long in waiting queues. However, users of the HPC system seem unconcerned about the longer times their jobs have to wait for execution because they tend to increase the estimate together with the size of their BoTs, shown by a positive correlation. To further our understanding of why HPC users tolerate the longer wait times, we calculate the occupation time and the estimated occupation time of BoTs[4]. Table 3.17 shows how much users utilize their estimates. Since HPC has the best utilization, HPC users seem to estimate their jobs better than users of other systems. Therefore, if HPC users decrease their estimates they may have the risk of underestimation, which can kill their jobs. To guarantee a successful execution for their jobs, they must tolerate longer wait times.

TABLE 3.16: Correlation between BoT sizes and BoT estimates.

FS0	FS2	FS3	HPC	LLN	NIK
-0.033	-0.244	-0.236	0.175	-	-0.145

[3]Since 65% jobs of LLN do not have the information about their user estimates, we decide to skip it when we analyze the BoT estimate.

[4]We define the occupation time of a job as the total time that it occupies processors, calculated as $R \times P$, where R and P are the runtime and the number of processors of the job, respectively. Similarly, the estimated occupation time of the job is $E \times P$, where E is the estimate of the job. The occupation time and the estimated occupation time of a BoT are the total occupation time and the total estimated occupation time of all jobs within the BoT.

TABLE 3.17: Utilization of user estimates, calculated by the ratio between occupation time per estimated occupation time of all BoTs.

FS0	FS2	FS3	HPC	LLN	NIK
4%	6%	4%	36%	-	16%

3.7 BoT Performance

This section deals with the performance issues of BoTs. It is interesting to know how much BoTs occupy processors. Table 3.18 shows the fractions of total occupation time of jobs belonging to BoTs per total occupation time of all jobs. As we can see from the table except for NIK, the occupation time of BoTs is not large despite the fact that BoTs are common in practice as shown in Section 3.4. In particular, the occupation time of BoTs in case of LLN is only 1%. The reason is that LLN has only 40% jobs submitted as part of BoTs and the average runtime of these jobs is relatively small, compared with all jobs (see Table 3.19). In contrast, the occupation time of BoTs in other traces is larger since jobs belonging to BoTs are more dominant (see Table 3.7) and the average runtime of these jobs is also larger (see Table 3.19).

TABLE 3.18: Fractions of occupation time by BoTs.

FS0	FS2	FS3	HPC	LLN	NIK
30%	20%	28%	50%	1%	81%

TABLE 3.19: Average runtimes of jobs belonging to BoTs (R1) and of all jobs (R2), in minutes.

	FS0	FS2	FS3	HPC	LLN	NIK
R1	6	5	4	311	2	408
R2	6	9	8	282	24	413

The next performance issue analyzed in this section is how long BoTs wait for execution. Table 3.20 shows fractions of wait times of BoTs, calculated by the ratio between total wait times of jobs belonging to BoTs per total wait times of all jobs. In case of NIK and HPC, most of the wait times belong to jobs that are submitted as part of BoTs. We particularly notice the case of HPC since its BoTs only occupy 50% the total occupation time (see Table 3.18) while they are responsible for 90% of the total wait time. Hence, we believe that more efficient scheduling algorithms for BoTs are essential to reduce the wait times of jobs belonging to BoTs.

TABLE 3.20: Fractions of wait time by BoTs.

FS0	FS2	FS3	HPC	LLN	NIK
16%	32%	58%	90%	39%	80%

3.8 Importance of Workload Features

In previous sections, we have shown the common existence of several statistical workload features in real parallel traces. In this section, we discuss the crucial roles of these features in performance issues of clusters, grids and clouds.

Capturing the behaviour of *Bag-of-Tasks* in modeling helps to study the impact of the temporal-spatial correlation on scheduling performance and to evaluate scheduling designs for BoT applications. With respect to *temporal burstiness*, a large gap with no job arrivals is an ideal time for resource sharing in grids. A good grid scheduler obviously can utilize this time to allocate grid jobs to a parallel system if the system is idle due to no job arrival. In this way, grid jobs will have a high chance to be executed sooner and the grid performance can be enhanced. Furthermore, a burst clearly may cause a severe congestion for a system when a large number of jobs enter the system in a short time. In this case, a public cloud can be utilized to extend the capacity of the congested system as suggested in [11], where the authors propose several scheduling and redirection strategies for a local cluster to determine when to borrow resources from a public cloud. For *long range dependence*, if job arrivals exhibit this feature, gaps will be longer and bursts will be larger. The longer gaps provide more time for resource sharing in grids while the larger bursts make the congestion issue more significant. For *spatial burstiness*, although bursts can cause a system to undergo periods of severe congestion, we would like to emphasise that the issue is even more severe when the bursts come with the spatial burstiness property. If jobs within a burst are short and small, the system will not be as severely congested as in the case when it undergoes bursts with big and long jobs. For *cross-correlation*, we demonstrated that different degrees of this feature can lead to inconsistent conclusions about the evaluation of scheduling performance [65].

A crucial perspective of workload characteristics is that they provide useful clues for schedulers to make better decisions. For example, the schedulers can take advantage of properties like LRD and temporal burstiness to know when and how long a system is overloaded in order to make suitable scheduling strategies. Finally, we have discussed the roles of workload features in performance issues of parallel and distributed systems and our discussion is supported by several performance studies [22, 39, 45, 46, 57, 65, 68, 69]. These studies indicate that most features on one hand have severe impacts on system performance. On the other hand, the interaction between many features can help to enhance scheduling performance in particular situations [65, 68]. However, the most important result from the studies is that designing

and evaluating scheduling algorithms without the workload features may result in inaccurate conclusions.

3.9 Summary

In this chapter, we provided a characterisation of parallel and grid workloads. The analysis does not focus on simple marginal distributions as in other workload characterisation studies [37, 38, 51, 63] but emphasizes statistical workload features. It firstly showed the common existence of long range dependence and temporal burstiness in grid and cluster job arrivals. Then, it characterised the common presence of temporal locality, cross-correlation and spatial burstiness of runtime and parallelism processes. The chapter also made an analysis of the Bag-of-Tasks behaviour. We identified several pattern structures of BoT arrivals. It turned out that the Generalized Pareto distribution is the best fit for BoT interarrivals. We also illustrated that BoT arrivals are bursty. We showed that BoT arrivals can have similar structures as job arrivals, but they can also be completely contrary, with respect to long range dependence and periodicity. In addition to BoT arrivals, statistics like autocorrelation and cross-correlation were applied to BoT sizes, runtimes, parallelisms and estimates. Furthermore, the performance problems of BoTs were also analyzed. Finally, the chapter discussed the importance of these workload features on clusters, grids and clouds.

Chapter 4

Modeling Job Arrivals

In this chapter, we present how to model job arrivals with long range dependence (LRD) and burstiness[1] characteristics. For network traffic, it is well-known that a Markov-modulated Poisson process (MMPP) [24, 35, 100] can capture these features. However, MMPPs are not suitable for modeling job arrivals. An MMPP-based model captures dependencies from data and introduces the dependencies into interarrival times. As we indicated in Section 3.2, job arrivals exhibit LRD under the representation of a rate process, but do not show LRD with the interarrival time representation, and so an MMPP-based model fails to produce LRD for job traffic. The limitation of MMPPs in modeling system job traffic is well studied in [46], which suggests that MMPPs should be used to capture short to middle-range autocorrelations. Li [46] indicates that LRD of job arrivals should be reliably rooted and modeled from a rate process. He introduced the multifractal wavelet model (MWM) [85] as a good choice when it comes to long-range autocorrelations. Also a detailed explanation of why MWM is trusted for modeling job traffic, instead of other models of network traffic like MMPPs, was provided. However, as MWM is applied to a rate process, capturing burstiness with the interarrival representation is an issue. In Section 4.2, we will show that MWM is not able to mimic burstiness of real data though it produces LRD well. Therefore, we modify MWM and adapt it to capture burstiness of interarrival times while it still keeps LRD in a rate process.

4.1 Theory of Multifractal Wavelet Model

This section provides a brief background of the theories on which MWM is developed. For more detailed information, we refer the reader to [85].

MWM treats a stochastic process via scaling techniques in order to decrease/increase its length to smaller/larger scales. At each scale j, MWM considers the stochastic process as a series of scaling coefficients, denoted by $\{c_j\}$. As we will clarify later

[1] The term "burstiness" used during this chapter refers to temporal burstiness for brevity.

in Section 4.3, the series of scaling coefficients at the highest scale is used as a rate process in modeling job arrivals. In addition, MWM also defines a series of wavelet coefficients at scale j, denoted by $\{d_j\}$, that correspond to $\{c_j\}$. The scaling and wavelet coefficients can be recursively computed by

$$c_{j,k} = \frac{1}{\sqrt{2}}(c_{j+1,2k} + c_{j+1,2k+1}), \tag{4.1}$$

$$d_{j,k} = \frac{1}{\sqrt{2}}(c_{j+1,2k} - c_{j+1,2k+1}), \tag{4.2}$$

$k = 0, \ldots, N_j - 1$, where N_j is the number of scaling/wavelet coefficients at scale j. By rearranging Eq. (4.1) and Eq. (4.2) to

$$c_{j+1,2k} = \frac{1}{\sqrt{2}}(c_{j,k} + d_{j,k}), \tag{4.3}$$

$$c_{j+1,2k+1} = \frac{1}{\sqrt{2}}(c_{j,k} - d_{j,k}), \tag{4.4}$$

Riedi et al. [85] found a simple constraint to guarantee the positivity of the scaling coefficients: $|d_{j,k}| \leq c_{j,k}$. Positivity is a desirable feature since rate processes are inherently non-negative. To satisfy this constraint, MWM defines the following multiplicative model

$$d_{j,k} = A_{j,k} \times c_{j,k}, \text{ with } A_{j,k} \in [-1, 1], \tag{4.5}$$

where $\{A_j\}$ is considered as a series of multipliers at scale j. At each scale j, MWM uses the symmetric beta distribution [79] to fit $\{A_j\}$. The variance of a random variable A of a symmetric beta distribution with the unique parameter p is given by

$$var(A) = \frac{1}{2p+1}. \tag{4.6}$$

MWM yields LRD for output processes by fixing the beta parameter at the smallest scale p_1 according to Eq. (4.6):

$$p_1 = \frac{1}{2var(A_1)} - \frac{1}{2}, \tag{4.7}$$

where $\{A_1\}$ is a series of multipliers at scale 1, and calculates the beta parameter at scale $j \geq 2$ recursively:

$$p_j = \frac{1}{2}\left[\frac{var(d_{j-1})}{var(d_j)}\,(p_{j-1}+1) - 1\right]. \tag{4.8}$$

For details about the mathematical demonstration of how Eq. (4.7) and Eq. (4.8) can help to produce LRD, readers are referred to [85].

TABLE 4.1: Quantifying the burstiness of real job arrivals and of those generated by MWM, expressed by the coefficient of variation C_v. Z indicates the percentages of zeroes in a rate process.

Trace	T (s)	900	1800	3600	7200
NOR	C_v data	33.14	33.14	33.14	33.14
	Z data	60%	52%	44%	37%
	C_v MWM	5.09	4.56	5.28	4.6
	Z MWM	0%	0%	0%	0%
NIK	C_v data	18.56	18.56	18.56	18.56
	Z data	50%	36%	19%	16%
	C_v MWM	4.34	4.55	4.06	4.28
	Z MWM	0%	0%	0%	0%

4.2 MWM and Burstiness

We apply MWM to traffic of a real grid and a real cluster to generate synthetic job arrivals and then use the coefficient of variation C_v to quantify burstiness. The results in Table 4.1 with different time scales T[2] show that MWM is not able to mimic burstiness of the real data. To find the cause, we do a direct observation to the real job arrivals. We analyze real rate processes and see that they contain a large number of zero values as shown in Table 4.1. Each value in a rate process represents the number of job arrivals per second in a time interval. Therefore, a zero value means that there is no arrival job in the corresponding time interval. The existence of a large number of zeroes in a rate process indicates the presence of temporal burstiness in the original job arrivals.

[2]In the context of parallel workloads, there is a range that should not be selected for a time scale. A too small time scale (e.g., less than 15 minutes) results in a large number of zeroes (more than 50%) in a rate process and this will affect LRD. A too large time scale (more than 3 hours) results in a relatively short rate process that is hard to apply MWM on.

Figure 4.1 gives an example to illustrate why MWM cannot produce burstiness as in real job arrivals. In this example, R_2, containing a large number of zeroes, is considered a real rate process with a time scale of 10 seconds. When applying MWM to R_2, we obtain a synthetic rate process like R_1 with no zero value. Because positivity is a feature of MWM, it can only produce values greater than 0 (we refer to Section 4.4 for a detailed explanation). Therefore, a rate process generated by MWM includes only nonzero values. This means that there exist job arrivals in all time intervals. In other words, long gaps with no arrivals are not found in a job arrival process generated by MWM. This is illustrated in Figure 4.1, where job arrivals 2 of the real data exhibit a long free gap and a large bursty time, but the synthetic job arrivals 1 do not. This fact contradicts the notion of burstiness where free gaps and tight bursts are required to occur. This contradiction together with the results in Table 4.1 give us the conclusion that MWM cannot capture burstiness of real job arrivals.

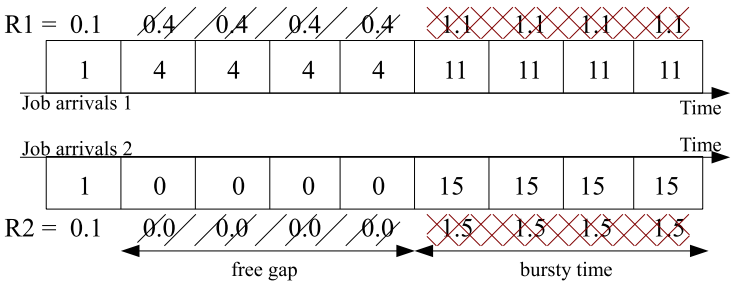


FIGURE 4.1: An example to illustrate why MWM does not capture burstiness. Each box represents a time duration of 10 seconds and the value in the box is the number of job arrivals in that duration.

4.3 Data Modeling and Synthesis of Standard MWM

MWM works through two fundamental procedures, so-called data modeling and synthesis. To model job traffic, the data modeling procedure will receive a real job arrival rate process as its input and train on the input to obtain a number of parameters. These parameters are then transferred to the synthesis procedure to generate a synthetic job arrival rate process.

The data modeling procedure is illustrated in Figure 4.2. This procedure represents the rate process input as a series of scaling coefficients $\{c_{j+1}\}$ at scale $j+1$. Starting with $\{c_{j+1}\}$, MWM uses Eq. (4.1) and Eq. (4.2) to calculate a series of scaling coefficients $\{c_j\}$ and a series of wavelet coefficients $\{d_j\}$ at scale j. Then $\{d_j\}$ is taken into account to calculate the beta parameter p_j according to Eq. (4.7) and Eq. (4.8). This procedure is repeated for smaller scales until the smallest scale ($j = 1$) is

reached. Finally, MWM produces the mean μ_c and the standard deviation σ_c of $\{c_1\}$ as well as the vector of beta parameters $\vec{p}$ whose components are the beta parameters calculated at each scale.

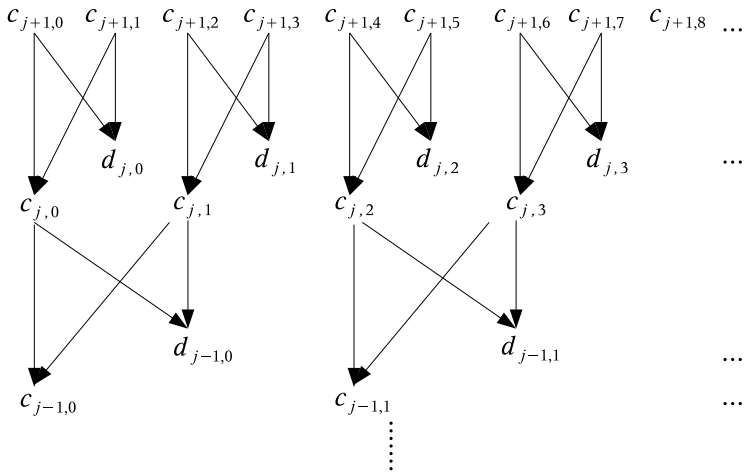


FIGURE 4.2: The data modeling procedure of MWM.

Taking μ_c, σ_c and $\vec{p}$ as inputs, the synthesis procedure in Figure 4.3 starts at the smallest scale by generating a series of N scaling coefficients $\{c_1\}$ as follows

$$c_{1,k} = \mu_c + \sigma_c \times randn, \text{ with } 0 \leq k \leq N-1, \tag{4.9}$$

where *randn* generates a random value that is normally distributed with mean 0 and standard deviation 1. Then the synthesis procedure yields $\{A_1\}$ using the beta distribution with the parameter p_1, the first component of $\vec{p}$. After that, a series of wavelet coefficients $\{d_1\}$ is obtained by Eq. (4.5) and is used together with $\{c_1\}$ to calculate scaling coefficients at the next higher scale $\{c_2\}$ with Eq. (4.3) and Eq. (4.4). This process is repeated for higher scales and is stopped after the last component of $\vec{p}$ is used. The final output of the synthesis procedure is a series of scaling coefficients at the highest scale $\{c_M\}$[3], which is used as a rate process in [46].

4.4 Modification of MWM

As shown in Section 4.2, MWM cannot capture burstiness in real job traffic because no zero value is found in its rate process. In this section we will show why MWM does not generate zeroes and how we can modify MWM to obtain zero values.

[3]From now on, M stands for max scale, meaning the highest scale.

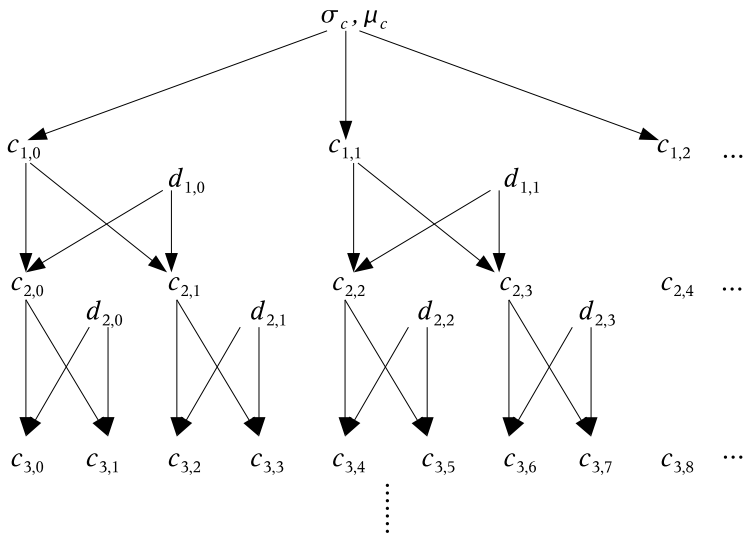


FIGURE 4.3: The synthesis procedure of MWM.

We answer these questions by two observations. First, we have from Eq. (4.3) and Eq. (4.4) that $c_{j+1,2k} = 0 \Leftrightarrow d_{j,k} = -c_{j,k}$ and $c_{j+1,2k+1} = 0 \Leftrightarrow d_{j,k} = c_{j,k}$. According to Eq. (4.5), these conditions are satisfied in case $A_{j,k} = -1$ or $A_{j,k} = 1$. However, we cannot obtain this with a beta distribution because its random variable is only distributed over the range $(-1, 1)$, and so all scaling coefficients $c_{j,k}$ are different from 0. This reason also explains why MWM has the strictly positive instead of non-negative feature as we desire. This shortcoming is also mentioned by the authors of MWM [85], where they note that the choice of a beta distribution for multipliers $A_{j,k}$ is not essential. Therefore, it should be dependent on the applied data to adapt the multipliers $A_{j,k}$ at each scale.

As a second observation, from Eq. (4.3), Eq. (4.4) and Eq. (4.5), we have $c_{j,k} = 0 \Rightarrow d_{j,k} = 0 \Rightarrow c_{j+1,2k} = c_{j+1,2k+1} = 0$. From this observation, we conclude that if we have, at a certain scale j, one scaling coefficient $c_{j,k}$ equal to 0, we will have at scale $j + n, 0 \leq n \leq M - j$, 2^n contiguous scaling coefficients equal to 0, as illustrated in Figure 4.4.

With this conclusion, we can control accurately the percentage of zero values in the series of scaling coefficients at the highest scale $\{c_M\}$ which is considered as a rate process. Let m be the number of scaling coefficients at scale j and let m_0 be the number of scaling coefficients at scale j that are equal to 0. Because each scaling coefficient $c_{i,k}$ generates two scaling coefficients $c_{i+1,2k}$ and $c_{i+1,2k+1}$, the number of scaling coefficients is $2m$ at scale $j + 1$, $4m$ at scale $j + 2$, and so on. We conclude that the numbers of scaling coefficients at scales $j, j + 1, ..., M$ are $m, 2m, ..., 2^{M-j}m$, respectively. Similarly, because each zero scaling coefficient generates two zero scaling

coefficients at the next scale as shown in Figure 4.4, we conclude that the numbers of zero scaling coefficients at scales $j, j+1, ..., M$ are $m_0, 2m_0, ..., 2^{M-j}m_0$, respectively. Consequently, the fractions of zero values at scales $j, j+1, ..., M$ are the same and equal to m_0/m. Thus, we can control the percentage of zero values in the final rate process represented by $\{c_M\}$ via handling scaling coefficients at any scale. However, we should not do this at scale 1 because $\{c_1\}$ is generated based on μ_c and σ_c to fit the marginal distribution. Rather, we should control the percentages of zero values in $\{c_2\}, \{c_3\}, ..., \{c_M\}$.

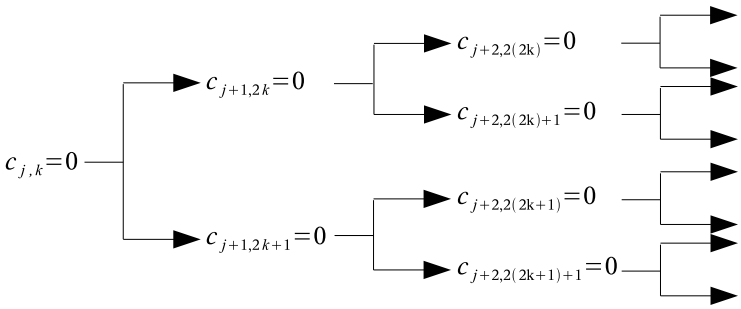


FIGURE 4.4: Generating zeros with modified MWM.

In addition to controlling the percentage of zero values in $\{c_M\}$, it is also important to control the way these zero values occur in $\{c_M\}$. For example, if real data contain 100 zero values that are consecutive, we should also produce 100 consecutive zero values. If we produce 100 zero values that are not consecutive, the idle period with no job arrival of synthetic data will be different from that of the real data and hence the produced burstiness is also different. In our study, this kind of distribution of zero values is controlled by calculating carefully the number of isolated zero values to be produced at each scale $j = 2, \ldots, M$. Each isolated zero value at a scale will expand by a factor of 2 at the next scale to a string of zeroes of the required length at $\{c_M\}$. In summary, we obtain $z\%$ zero values in the final rate process by generating $z_2\%$ isolated zero values in $\{c_2\}$, $z_3\%$ isolated zero values in $\{c_3\}$,..., and $z_M\%$ isolated zero values in $\{c_M\}$ in such a way that $z = \sum_{j=2}^{M} z_j$. Now, we need to answer the question of how to calculate z_j for $j = 2, ..., M$ so that we achieve a distribution of zero values in $\{c_M\}$ similar as in real data. As we already found above, if at a certain scale j we have $c_{j,k} = 0$, we will at scale $j+n$ have 2^n contiguous scaling coefficients with value 0. In other words, for each contiguous series of r zero value scaling coefficients in $\{c_M\}$, we deduce that there is an isolated zero value in $\{c_j\}$, where $j = M - \log_2 r$. Based on this idea, we propose the following steps to calculate z_j:

1. Calculate the zero percentage z in the real job rate process $\{data\}$.
2. Compute the lengths of the maximal contiguous sequences of zeroes in $\{data\}$ to obtain a series $\{r\}$. For example, if $\{data\} = \{0.01, 0.02, 0, 0, 0, 0, 0.09, 0, 0, 0, 0.02, 0.03, 0\}$, then $\{r\} = \{4, 3, 1\}$.

3. $\forall r_k \in \{r\}$, let $s_k = M - \log_2 r_k$ to obtain a series of scales $\{s\}$.
4. For each $j = 2$ to M, let $z_j = f_j \times z$, where f_j is the fraction of occurences of j in $\{s\}$.

Now we have determined the fraction of zero values to be generated at each scale, but there is still the question of how to do the generation. In order to obtain $z_j\%$ zero values in $\{c_j\}$, we will generate $\{A_{j-1}\}$ with $length(\{c_j\}) \times z_j$ values 1 or -1: if $A_{j-1,k} = 1 \Rightarrow c_{j,2k+1} = 0$, otherwise $A_{j-1,k} = -1 \Rightarrow c_{j,2k} = 0$.

Although our method in theory can assure the percentage of zero values in a synthetic workload to be exactly the same as in a real trace, they can be slightly different in practice for the following reason. If we have $length(\{c_1\}) = m$, the length of the synthetic workload is $m \times 2^{M-1}$. In practice, the length of the real trace does not have a so-called "power-of-two-like" shape. Rather, its length is cut off in the data modeling procedure as well as in the first of the four steps above to get a "power-of-two-like" shape. For example, if the length of the real trace is 52, it is cut off to $48 = 3 \times 2^4$.

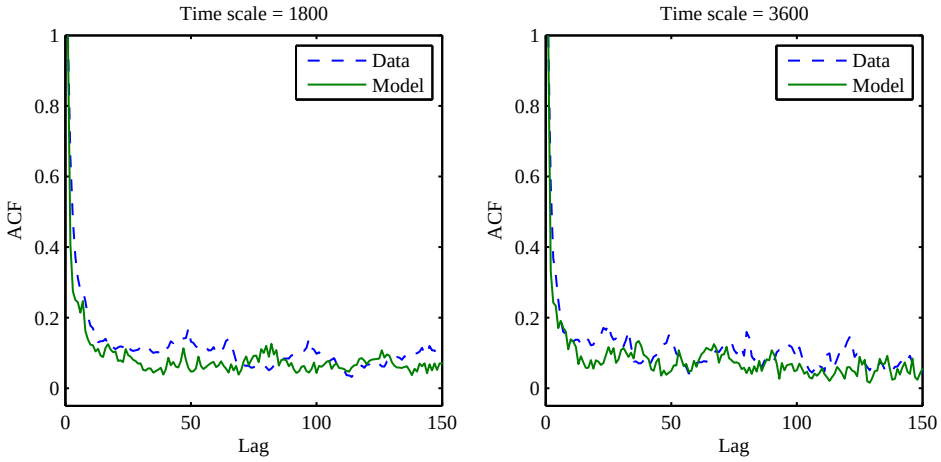


FIGURE 4.5: The autocorrelation functions of real and synthetic rate processes in case of LAN with different time scales.

4.5 Experimental Results

We will present in this section our experiments to evaluate the quality of our job arrival model. The evaluation is done by comparing the synthetic job arrivals generated by the model with those in real traces. Workload features including long range dependence and temporal burstiness as well as the marginal distribution of job interarrival times are taken into evaluation.

4.5.1 Long Range Dependence

Since we represent job arrivals by a rate process which depends on a time scale T, we consider several time scales in this experiment to compare our model with the data of a grid (NOR) and a cluster (LAN). The LRD of a stochastic process can be visually observed via its autocorrelation function (ACF). We draw in Figure 4.5 the ACFs of the rate processes of LAN and the synthetic rate processes generated by our model. Furthermore, we also quantify LRD by estimating the Hurst parameter with the estimate approach described in Section 3.2 and show results in Table 4.2. From Figure 4.5 and Table 4.2, we see that our job arrival model gives a good result because LRD is controlled well by Eq. (4.7) and Eq. (4.8). Moreover, the model is also stable over different time scales.

TABLE 4.2: LRD of job arrivals via estimating the Hurst parameter.

T (s)	900	1800	3600	7200
NOR	0.86 ± 0.07	0.89 ± 0.07	0.88 ± 0.08	0.84 ± 0.08
Model	0.79 ± 0.06	0.87 ± 0.14	0.85 ± 0.14	0.81 ± 0.11
LAN	0.80 ± 0.03	0.81 ± 0.03	0.78 ± 0.01	0.81 ± 0.02
Model	0.81 ± 0.05	0.87 ± 0.16	0.83 ± 0.11	0.78 ± 0.12

TABLE 4.3: The coefficient of variation of interarrival processes.

Time scale (s)	900	1800	3600	7200
NOR	33.14	33.14	33.14	33.14
MWM	5.09	4.56	5.28	4.60
Model	24.65	25.81	25.77	27.20
LAN	17.70	17.70	17.70	17.70
MWM	2.93	2.48	2.24	2.15
Model	13.32	13.79	13.74	13.74

4.5.2 Temporal Burstiness

We measure temporal burstiness based on calculating the coefficient of variation of an interarrival process, which is converted from a rate process using the *integrate-and-fire* algorithm [107]. Calculated results in Table 4.3 confirm that our model controls

temporal burstiness better than MWM. In addition, we also show the complementary cumulative distribution functions (CCDFs) of the real and synthetic interarrival processes. As we can see from Figure 4.6, our job arrival model is able to capture temporal burstiness well since its interarrival times are fitted nicely to those of the real data.

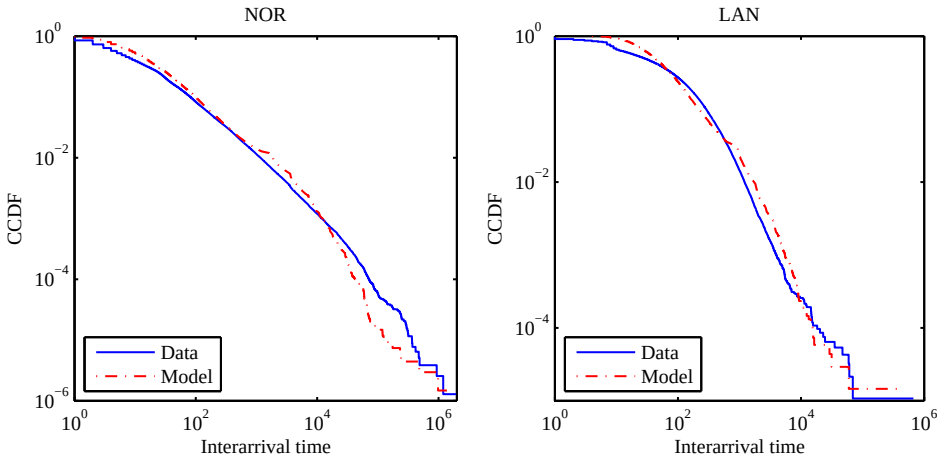


FIGURE 4.6: The complementary cumulative distribution functions of the interarrival times of NOR, LAN and the model.

4.6 Summary

Job traffic plays a crucial part in workloads of parallel systems and grids. Therefore, modeling job arrivals is essential to provide realistic workloads for study on performance evaluation of scheduling. Towards this end, this chapter introduced a new job arrival model that can capture two observed features of real job traffic, namely long range dependence and temporal burstiness. The model was developed based on the multifractal wavelet model [46, 85]. Experimental results showed that the developed job arrival model can accurately control the temporal burstiness degree and can produce long range dependence well. Moreover, the synthetic job arrival process generated by the model also fits the marginal distribution nicely. However, the daily cycle feature is a shortcoming of this job arrival model which should be investigated in future work.

Chapter 5

A Comprehensive Parallel Workload Model

In this chapter, we present a full and realistic workload model of parallel systems that captures most of the characteristics mentioned in Section 2.3, including long range dependence and temporal burstiness of job arrivals; spatial burstiness and correlation of runtimes and parallelisms; and the Bag-of-Tasks behaviour. Our model takes a real workload W as its input and generates a synthetic workload W' which is similar to W with respect to these characteristics. Because we consider a parallel workload as consisting of triples {arrival time, runtime, number of processors}, we generate W' as including an arrival process $\{Arr_i\}$, a runtime process $\{Run_i\}$, and a parallelism process $\{Cpu_i\}$. Since the synthetic arrival process $\{Arr_i\}$ with LRD and with temporal burstiness is obtained with the job arrival model in Chapter 4, we only focus in this chapter on how to generate $\{Run_i\}$ and $\{Cpu_i\}$ in such a way that we can control the Bag-of-Tasks behaviour, the spatial burstiness, and the correlation between $\{Run_i\}$ and $\{Cpu_i\}$.

5.1 The Comprehensive Model

Generating a workload with our model consists of four stages. The first and second stages are to classify the runtime and the parallelism processes, respectively. At the next stage, we will fit BoT sizes to a Zipf distribution as shown in Section 3.4. The outputs of the first three stages together with $\{Arr_i\}$ serve as inputs of the final stage, which is the main algorithm of the model.

5.1.1 Runtime Classification

As presented in Section 3.3.1, runtime classification is done by using Model-Based Clustering (MBC) [28] in our study. The input of the runtime classification procedure

is a runtime process $R_i, i = 1, \ldots, n$ obtained from W. The procedure runs MBC to classify $\{R_i\}$[1] and obtain *mixture of Gaussians* parameters $(\mu_k, \sum_k; p_k), k = 1, \ldots, G$ and classification labels $L_i \in \{1, \ldots, G\}, i = 1, \ldots, n$, where G is the number of clusters, L_i represents the cluster to which R_i belongs, and μ_k, $\sum_k$ and p_k are the mean, the variance, and the probability of cluster k, respectively.

5.1.2 Performance of BoTs

As we indicated in Section 3.7, although BoTs are common, their occupation time is small, compared to the overall occupation time of all jobs. For example, the occupation time of BoTs in case of LLN is only 1%. The reason is that users must restrict the runtimes of their jobs when they submit the jobs as BoTs. This is a reasonable behaviour to avoid a long wait time. Therefore, we argue that long runtimes should not be associated with a BoT. In other words, the runtime distributions of jobs belonging to BoTs and single jobs should be different. We calculate these distributions from the probabilities $\{p_k\}$ obtained in Section 5.1.1. For each runtime cluster $k \in [1, G]$, we calculate the probability that k is associated with a single job (a_k) or a job of a BoT $(1 - a_k)$, thus p_k is divided into two parts: $p_k = a_k p_k + (1 - a_k)p_k$. Finally, we obtain the runtime distribution for single jobs as $\{pjob_k = a_k p_k / \Sigma_{i=1}^{G} a_i p_i\}$ and the runtime distribution for jobs of BoTs as $\{pbot_k = (1 - a_k)p_k / \Sigma_{i=1}^{G}(1 - a_i)p_i\}$.

5.1.3 Parallelism Classification

The input of the parallelism classification procedure is a parallelism process $Par_i, i = 1, \ldots, n$ obtained from W. The procedure will classify $\{Par_i\}$ to obtain classification labels $\{C_i\}$, where C_i represents the class to which Par_i belongs. Our approach to classify the parallelism is as follows. We start by grouping jobs that require the same number of processors and count the number of jobs in each group. If the classification procedure stops here, we may have a large number of groups. For example, if the system where we collected the workload has 4096 processors (as in case of the trace LLN), we may obtain 4096 groups with this classification method. Since we use a transition conditional probability table $Pr(c, l)$ to control the cross-correlation between the runtime and the parallelism as presented later in Section 5.1.4, we need to reduce this large number of groups to avoid the problem of overfitting. $Pr(c, l)$, where c and l are labels of the parallelism and the runtime, respectively, indicates the probability for a job to have parallelism label c with the condition that the label for its runtime is known in advance as l. If the number of groups is large, the size of the table $Pr(c, l)$ increases and it leads to the problem of overfitting. Hence to reduce the number of groups, we assign each group a label which is the integer calculated by rounding the logarithm of the number of jobs in that group to base 2 and

[1] We consider denotations $X_i, i = 1, \ldots, n$ and $\{X_i\}$ equivalent to indicate a stochastic point process.

adding 1. Jobs belonging to the same group will be classified with their group label. As such, groups that have approximately equal numbers of jobs will be aggregated and be assigned the same label. For example, if there are 250 jobs requesting 4 cpus and 300 jobs requesting 10 cpus, all of them will be classified as 9. This method significantly reduces the number of groups, as for the case of LLN: from potentially 4096 down to 17 groups. The reason we aggregate equal-size groups is that when we convert a parallelism label to a specific number of processors, we simply use the uniform probability as shown later in Algorithm 2.

In summary, the main idea of the parallelism classification procedure is first grouping jobs that require the same number of processors, and then aggregating groups that have approximately equal numbers of jobs to form new groups. The aggregation of equal-size groups is for two purposes: to reduce the number of groups avoiding the problem of overfitting and to simplify the conversion from a parallelism label to a specific value.

5.1.4 The Complete Model

Given a job arrival process $\{Arr_i\}$ obtained with the model in Chapter 4, we summarize our full model as the following stages:

1. Call the runtime classification procedure described in Section 5.1.1 to obtain *mixture of Gaussians* parameters $(\mu_k, \sum_k; p_k), k = 1, \ldots, G$ and classification labels $L_i \in \{1, \ldots, G\}, i = 1, \ldots, n$. Then calculate the runtime distributions of single jobs $\{pjob_k\}$ and jobs of BoTs $\{pbot_k\}$ as presented in Section 5.1.2.
2. Call the procedure in Section 5.1.3 to classify the parallelism process and determine classification labels $C_i, i = 1, \ldots, n$.
3. Fit the BoT sizes from the real data to a Zipf distribution Z.
4. Call Algorithm 1 with inputs from the above steps to obtain a synthetic runtime process $\{Run_i\}$ and a synthetic parallelism process $\{Cpu_i\}$. The set of triples $\{Arr_i, Run_i, Cpu_i\}$ constitutes the full synthetic parallel system workload W'.

In Algorithm 1, we first calculate the transition conditional probability table $Pr(c, l)$ mentioned in Section 5.1.3. $Pr(c, l)$ of a job is calculated by the ratio between the probability $P(c, l)$ for that job to have parallelism label c and runtime label l at the same time and the probability $P(l)$ for that job to have runtime label l. Note that, for a real workload, we only need to calculate the table $Pr(c, l)$ once and apply it several times to generate several synthetic workloads. Secondly, we initialize the runtime Run_1 and the number of processors Cpu_1 for the first job in the synthetic workload by calling the function RandomlyGenerate. This function will generate randomly a pair of runtime and number of processors in such a way that we can control

Algorithm 1 Generate synthetic runtimes and parallelisms. n is the number of jobs in the real data. The model enables the generation of as many jobs as desired.

Input: Job arrivals $\{Arr_i\}$, clustering parameters $(\mu_k, \sum_k; pjob_k, pbot_k), k = 1, \ldots, G$, runtime classification $L_i, i = 1, \ldots, n$, parallelism classification $C_i, i = 1, \ldots, n$, the BoT parameter Δ and the fitted BoT size Zipf distribution Z.
Output: A synthetic runtime process $\{Run_i\}$ and a synthetic parallelism process $\{Cpu_i\}$.

// Calculate the transition conditional probability table
$maxL = max(\{L_i\}) = G$; $maxC = max(\{C_i\})$;
for $l = 1$ to $maxL$ **do**
  $P(l) = \frac{length(\{x=l, x \in \{L_i\}\})}{n}$;
  for $c = 1$ to $maxC$ **do**
    $P(c,l) = \frac{length(\{j \in [1,n]: C_j = c, L_j = l\})}{n}$, where j represents a job;
    $Pr(c,l) = \frac{P(c,l)}{P(l)}$;
  end for
end for

// Initialize
Assign $BoTs = 1$ and sample $BoTSize$ from the fitted Zipf distribution Z;
$[Run_1, Cpu_1]$ = RandomlyGenerate($BoTSize$);

// Main loop to generate $\{Run_i\}$ and $\{Cpu_i\}$
for $j = 2$ to $length(\{Arr_i\})$ **do**
  if ($Arr_j - Arr_{j-1} \leq \Delta$ *and* $BoTs < BoTSize$) **then**
    $Run_j = Run_{j-1}$;
    $Cpu_j = Cpu_{j-1}$;
    $BoTs = BoTs + 1$;
  else
    Assign $BoTs = 1$ and sample $BoTSize$ from the fitted Zipf distribution Z;
    $[Run_j, Cpu_j]$ = RandomlyGenerate($BoTSize$);
  end if
end for

function $[rRun, rCpu]$ = RandomlyGenerate(bs)
1. Randomly select a runtime label $sl \in [1, G]$ using probabilities $\{pbot_k\}$ if $bs > 1$ or $\{pjob_k\}$ if $bs = 1$;
2. Randomly select a parallelism label sc using the transition probability table $Pr(c,l)$ with $l = sl$;
3. Assign $rRun$ by sampling the Gaussian distribution $f_{sl}(\mu_{sl}, \sum_{sl})$;
4. Assign $rCpu$ by calling Algorithm 2 with inputs sl and sc;

end

Algorithm 2 Generate the synthetic parallelism for a job.

Input: Runtime classification $L_i, i = 1, \ldots, n$, parallelism classification $C_i, i = 1, \ldots, n$, runtime label sl and parallelism label sc.

Output: Number of processors $procs$.

1. Determine all jobs in the real data whose runtime and parallelism labels are sl and sc, respectively based on $\{L_i\}$ and $\{C_i\}$. Let X be the multiset, i.e. including multiple occurrences, of the numbers of processors of these jobs.

2. Select uniformly at random an element of X to obtain $procs$.

their cross-correlation. This cross-correlation is indeed controlled by steps 2 and 4 in the function since the parallelism label sc is selected using the transition conditional probability table $Pr(c, l)$ where the runtime label sl is already known in advance. With the selected label sc, the parallelism value is generated using Algorithm 2. This algorithm determines all jobs in the real data with runtime label sl and parallelism label sc and forms a multiset of the numbers of processors of these jobs. Then, it generates the parallelism value by selecting uniformly at random an element of the multiset. Thirdly, we control BoT behaviour in the main loop. For any two consecutive jobs $j - 1$ and j that satisfy the condition $Arr_j - Arr_{j-1} \leq \Delta$[2], we consider them to be similar and thus they have the same runtime and number of processors. In addition, we also control the size of each BoT by sampling a value $BoTSize$ from the fitted Zipf distribution Z. Whenever the size of a BoT reaches $BoTSize$, we will stop that BoT and form a new BoT by calling the function and sampling a new value for $BoTSize$.

As for the complexity of the two algorithms, it is easy to see that the complexity of Algorithm 2 is $O(n)$ since we have to traverse $L_i, i = 1, \ldots, n$ and $C_i, i = 1, \ldots, n$ in the first step of this algorithm. With respect to Algorithm 1, we do not count the step of calculating the table $Pr(c, l)$ in its complexity because this table is static, i.e. calculated once. Algorithm 1 takes $O(m-1)$ times for the main loop, where m is the number of synthetic jobs, and the loop calls Algorithm 2 inside. Therefore, the total complexity of Algorithm 1 is $O(m.n)$.

5.2 Experimental Results

We will present in this section our experiments to validate our model. We apply our model to real traces to generate synthetic workloads. The quality of the synthetic workloads is evaluated by comparing them with the real data. Long range dependence and temporal burstiness properties of the synthetic job arrival process are controlled well by our job arrival model in Chapter 4. In this section, we evaluate the Bag-of-Tasks behaviour, spatial burstiness and the cross-correlation between runtime and

[2]The parameter Δ is used to form a BoT as described in Section 2.3.3.

parallelism. Quantitative metrics for these characteristics are given in Section 2.3. In addition, we also evaluate our model based on the marginal distributions. Finally, we present a simulation experiment with our model and compare its performance with that of real world data.

5.2.1 Bag-of-Tasks Behaviour

In our experiments, we consider two aspects of BoT behaviour. We first would like to know how well BoT sizes are distributed comparing with real data. Secondly, we evaluate the marginal distribution of BoT runtimes. The runtime of a BoT is calculated as the average of the runtimes of all jobs within the BoT. The complementary cumulative distribution functions of both BoT sizes and BoT runtimes are shown in Figure 5.1. It can be seen that our model nicely fits the real data. Note that for BoT runtimes, we only consider "real" BoTs whose sizes are greater than 1, because they are the main target of our model. If we include "unreal" BoTs, i.e., whose sizes are equal to 1, in the figure, it is impossible to visually differentiate between "real" and "unreal" BoTs.

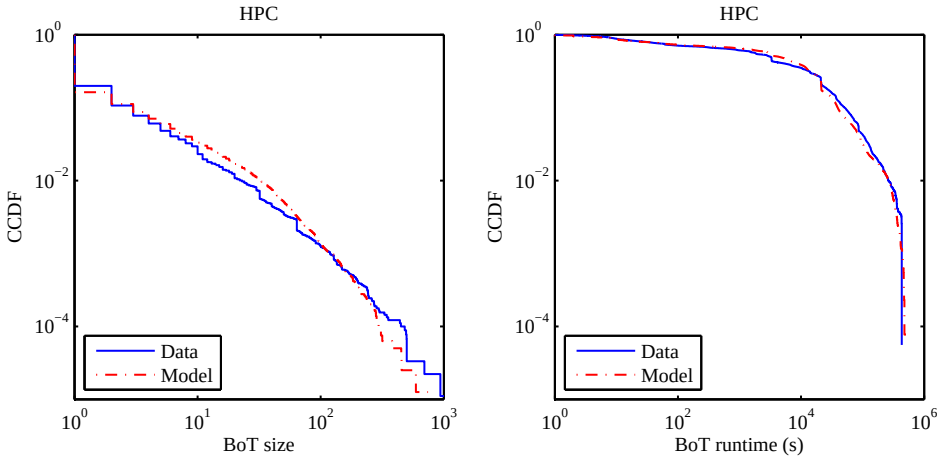


FIGURE 5.1: The complementary cumulative distribution functions of the BoT sizes and the BoT runtimes of HPC and the model.

5.2.2 Spatial Burstiness

To evaluate how well the approach of using normalized entropy proposed in Section 2.3.2 works on measuring spatial burstiness, we compare our model with real data and with a naive model such as the uniform distribution since it is commonly used in practice. The fact is that a naive model cannot capture spatial burstiness, thus

results of applying the entropy approach on this kind of model should reach 1. In our experiment, the naive model will use the uniform distribution to generate runtimes and parallelisms. It can be seen from Table 5.1 that our model captures well the spatial burstiness.

TABLE 5.1: The normalized entropy of spatial burstiness.

	FS2	FS3	HPC	LAN	LLN
Data	0.447	0.126	0.305	0.317	0.330
Our Model	0.536	0.192	0.316	0.308	0.320
Naive Model	0.998	0.998	0.999	0.997	0.996

5.2.3 Marginal Distributions

Another important result from our model is that the marginal distributions of the runtime and the parallelism match well, as can be seen in Figure 5.2. For the runtime with continuous values, the marginal distribution is determined by the *mixture of Gaussians* model (see Section 5.1.1). For the parallelism with discrete values, our experiments prove that the marginal distribution is matched well by the combination of the parallelism classification procedure in Section 5.1.3 and the transition probability table defined in Section 5.1.4.

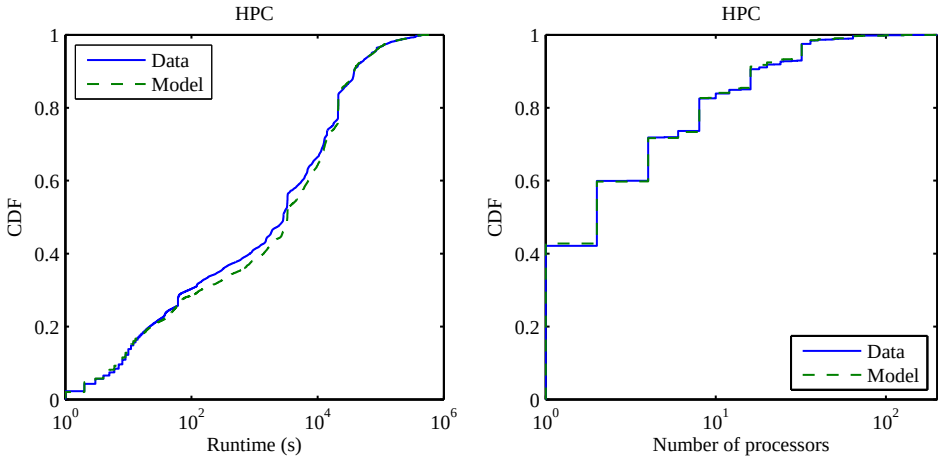


FIGURE 5.2: The cumulative distribution functions of the runtimes and the parallelisms of HPC and the model.

5.2.4 Correlation between Runtime and Parallelism

One of the most difficult problems in modeling parallel workloads is how to control the (cross-)correlation between runtime and parallelism as in real data. The correlation is measured by calculating the Spearman correlation coefficient between the two attributes. It can be seen from Table 5.2 that our model controls the correlation well since our results are close to the real data. The correlation is controlled well thanks to the combination of the transition probability table defined in Section 5.1.4 and the way we generate specific values for parallelism labels in Algorithm 2.

TABLE 5.2: The correlation between runtime and parallelism.

	FS2	FS3	HPC	LAN	LLN
Data	+0.459	−0.247	−0.229	+0.073	+0.453
Model	+0.389	−0.285	−0.205	+0.088	+0.457

5.2.5 Simulation-Based Evaluation

This section presents how good our model is in the simulation of parallel system scheduling. In our experiment, we apply the model to the LLN trace to generate 10 synthetic workloads. Then we use all the workloads including the original trace in the simulation of a parallel system scheduler and compare their scheduling performance. We build our simulation environment on GridSim [5] and implement two scheduling policies in our study, namely first-in first-out (FIFO) and EASY [73]. The performance metric we select is *Slowdown*, which is used as a function of *System Load*. *System Load* is defined by $\lambda \times E[runtime \times jobsize]/N$, where λ is the arrival rate, N is the system size (total number of processors), and $E[\cdot]$ is the average value. *Slowdown* is defined as the average job response time divided by the average job runtime. This metric is not the same as average slowdown. The difference and the reason to select this metric are given in [57]. Each workload in this experiment contains $\sim 100K$ jobs, so we simulate around 1 million jobs from the LLN trace and the 10 generated workloads.

The simulation results are shown in Figure 5.3. As we can see for both policies, the scheduling performance quickly decreases (larger slowdown) when the system load approaches saturation. Furthermore, the scheduling performance of the trace is covered by that of the synthetic workloads and is close to the average result. Note that, it is impossible to expect that every synthetic workload always produces the same performance with the trace, even if the workload is generated by a perfect model because changing the runtime of one job by only 30 seconds can result in a change of 8% in parallel scheduling performance [106]. Therefore, it is important that the scheduling performance of synthetic workloads should cover that of a trace. This means if a

new scheduling algorithm is proved to work well with the synthetic workloads, we can believe that it also works well with the trace. It is also crucial that the trace produces a performance close to the average performance of the synthetic workloads because it helps to quantitatively evaluate a scheduling algorithm. Therefore, the experimental results indicate that our model is representative enough for a real workload for simulation use.

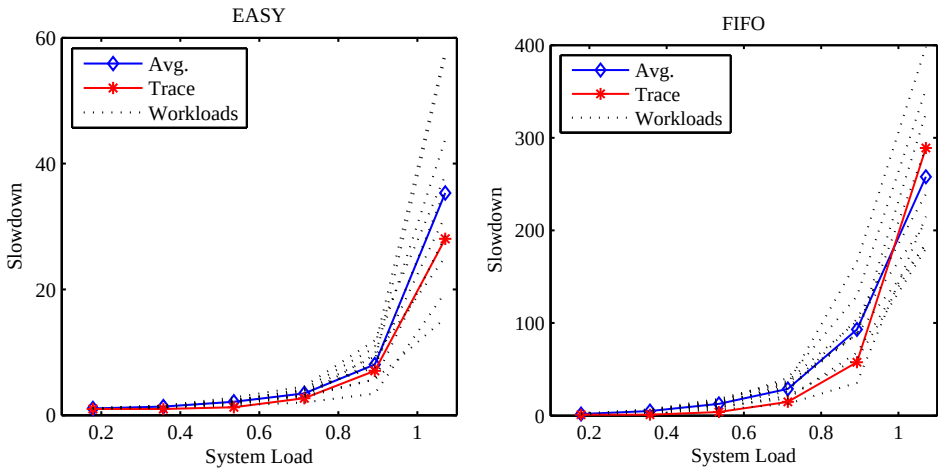


FIGURE 5.3: The scheduling performance of LLN and the synthetic workloads generated by our model. Each dotted line represents a synthetic workload and the Avg. is averaged from the dotted lines. System load is varied by adjusting the system size.

5.3 Related Work

The quality of a workload model highly depends on its representativeness for real world data because performance evaluation follows the garbage-in-garbage-out principle: using unrealistic workloads as input will yield inaccurate results. A workload model that is able to capture the marginal distributions and all workload properties observed in real traces will be the most representative model since it generates workloads which resemble those that a system faces. However, it is not easy to create such a perfect model. Although many statistical models have been proposed [8, 18, 39, 40, 53, 54, 59, 67, 95], all of them can only capture one or two workload characteristics at the same time and neglect the other properties.

The only model for BoTs we found is the one recently proposed by Iosup et al. [39]. However, their model is mainly for grid workloads and is not suitable for parallel workloads because they assume that all jobs are serial. Song et al. [95] developed a parallel workload model based on Markov chains [41] to capture the correlation. Li et al. [53] suggested a model for job arrivals that is able to produce short autocorrelation

structures. They also created a model [54] for job runtimes, where parallelism is not taken into account. Job runtimes generated by this model exhibit locality as in [18]. Since runtimes within a BoT are similarly repeated, our model produces locality by default when it generates BoTs. Another model that is able to yield both locality and correlation is proposed in [67]. The model in [40] generates job arrivals and runtimes based on fitting real data to some marginal distributions but does not concentrate on any workload property. A positive common point of all these models is that the marginal distributions can be fitted to real data. Their drawback is that they can only capture at most two workload properties. In addition, these models also have another significant drawback since they only generate either one [18, 53, 54] or two [39, 40, 67, 95] attributes of parallel workloads.

Typically, a parallel workload includes three attributes: arrival time, runtime and parallelism. The only two models that are able to generate all three attributes like our model are [8] and [59]. However, the former [8] only focuses on fitting marginal distributions and non of the workload features is modeled. The model introduced by Lublin and Feitelson [59] can produce neither LRD, temporal and spatial burstiness nor BoTs, except for the cross-correlation and the daily cycle feature. This model is developed mainly on the base of fitting real data to marginal distributions. Moreover, since job arrivals are generated separately from job runtimes and parallelisms, the temporal-spatial correlation is not taken into account in the model.

The shortcomings of current models make them less representative. Using representativeness as a criterion, we believe that our model is more representative than those models since it can capture most of observed workload properties including the marginal distributions.

5.4 Summary

We analyzed in Chapter 3 several grid and parallel system traces with respect to five characteristics, namely long range dependence and temporal burstiness of job arrivals, Bag-of-Tasks behaviour, and spatial burstiness and correlation of runtime and number of processors. In Chapter 3, we showed that these features exist commonly in real world data. We also discussed in Section 3.8 with arguments supported by many performance studies [39, 45, 46, 57, 65, 68, 69], that they have significant impacts on the performance of clusters and grids. Therefore, we concluded that a representative workload model should be able to capture all these features. We then developed such a model in this chapter. Our model is validated by fitting and comparing with real data, a validation method that is widely accepted in the research community of system workload modeling [8, 53, 59, 95]. In addition, we also did an experiment to show that the model is representative enough for simulation use.

The model can be used in a more flexible way by changing the parameter Δ in the BoT definition and the parameter of the Zipf distribution to adjust the Bag-of-Tasks

behaviour. We refer readers to [65, 68, 69] for the practical use of our model. These studies apply the model to illustrate the important roles of the features and to show their real performance impacts on clusters and grids. Furthermore through the model, the studies also provide some useful clues to enhance scheduling performance.

Currently, our model does not support periodicity such as daily cycles and BoTs cannot be interleaved with each other. These issues are left as future work.

Chapter 6

Performance Impact of Job Arrivals on Clusters and Grids

Workloads where job arrivals play a vital role are an indispensable part during the performance evaluation process of parallel and distributed schedulers. Therefore, job arrivals are considered as one of the key factors that can affect the reliability of the evaluation results. In Chapter 3, we have shown that real cluster and grid job arrivals exhibit two important characteristics, namely long range dependence (LRD) and burstiness[1]. Using workloads with/without these two characteristics in an evaluation can lead to incorrect conclusions since they may potentially have severe impacts on scheduling performance. In this chapter, we study how LRD and burstiness structures of job arrivals in a workload affect the performance of clusters and grids through realistic model-based simulations. The representative statistical job arrival model proposed in Chapter 4 is used to generate realistic workloads with these characteristics, which are then run by means of simulation to obtain dependable results.

6.1 Workload Models

Three patterns of job arrivals are created in our study. Firstly, a pattern that exhibits neither dependence nor burstiness. Secondly, one that contains long range dependence. Thirdly, both dependence and burstiness are present in the job arrivals. We denote workloads with these three patterns as b_N, b_L, b_{LB} in case of a background workload and g_N, g_L, g_{LB} in case of a grid workload, respectively in later sections. The first pattern of job arrivals is generated with the Poisson distribution because Poisson arrivals are not dependent [49]. Furthermore, since Poisson interarrivals are approximately equal, hence Poisson arrivals are not bursty. With respect to dependence, the multifractal wavelet model (MWM) [85] is applied to fit rate processes because it is shown that LRD can be well reconstructed by MWM [49]. Although MWM is

[1]The term "burstiness" used in this chapter refers to the feature temporal burstiness.

a good choice to produce dependence for a stochastic process, Chapter 4 shows that applying MWM cannot capture burstiness of real data. Therefore, we proposed in Chapter 4 a modified version of MWM, the so-called MMWM, to capture burstiness. MMWM is a job arrival model that does not only support LRD and burstiness, but also fits the marginal distribution well. As such, with Poisson, MWM and MMWM, all three patterns of job arrivals necessary for our experiments are achieved. Runtime and requested number of processors of a job are generated uniformly to ensure no potential characteristic of runtimes and jobsizes affects our evaluation results because the feature temporal locality of runtimes and the correlation between runtime and parallelism have significant impacts on parallel system performance as is shown later in Chapter 7.

6.2 Realistic Simulation and Requirements

In our study, the simulation environment is built on GridSim [5]. GridSim is a discrete-event based simulator which provides core entities of clusters and grids such as jobs, resources and information services. The performance metric we select is *Slowdown*, which is used as a function of either *System Load* or *Grid Load*. *Slowdown* and *System Load* are defined in Section 5.2.5. *Grid Load* is the load of a grid consisting of n clusters. We calculate the number of free processors on a cluster i as $(1 - L_i) \times N_i$, where N_i and L_i denote the size and the load of the cluster, respectively. We define *Grid Load* by $\lambda_g \times E[runtime_g \times jobsize_g]/N_g$, where λ_g is the arrival rate of grid jobs, N_g is the total number of free processors of all clusters in the grid, and $E[\cdot]$ is the average value of the products between runtimes and jobsizes of grid jobs.

TABLE 6.1: Details of The Distributed ASCI Supercomputer 3.

Cluster	Location	Node	Type	Speed	Memory
VU	Vrije University	85 dual	dual-core	2.4 GHz	4 GB
LU	Leiden University	32 dual	single-core	2.6 GHz	4 GB
UvA	Amsterdam University	41 dual	dual-core	2.2 GHz	4 GB
TUD	Delft University	68 dual	single-core	2.4 GHz	4 GB
UvAMN	MultimediaN	46 dual	single-core	2.4 GHz	4 GB

To evaluate the performance of a system, we cannot just evaluate one workload and draw conclusions. This will be inaccurate because the evaluation result is only correct for that workload and cannot be generalized. Therefore, we evaluate many workloads and base our conclusions on the average result. In our study, each data point in the performance results, shown by graphs in later sections, is calculated as the average of 25 (grid scenario) to 50 (cluster scenario) simulation runs with

25 to 50 different workloads. The number of jobs in a workload is also important for parallel job scheduling evaluation. Long workloads, i.e., containing thousands of jobs as suggested in [25], are not only necessary to achieve a steady state, but even more to see the realistic conditions that a real system faces. Each workload in our study contains over 63K jobs and hence our experimental workloads satisfy the size requirement for a reliable simulation of schedulers. For a total of 150 workloads, we have to simulate ∼10 million jobs in our experiments.

6.3 System Scenarios

To evaluate the performance impacts of LRD and burstiness, we create two system scenarios in our experiments, one for a cluster and one for a grid. In the realistic scenario of a single cluster, we implement three scheduling policies, namely FIFO, Aggressive and Conservative backfilling [73]. They are the most common scheduling policies used in real clusters nowadays. Investigating the Parallel Workloads Archive website [80], we see that most clusters implement FIFO with support of backfilling. FIFO schedules jobs on a first come first served base. Backfilling means that short jobs which are expected to terminate in time are allowed to leapfrog and execute before previously queued large jobs with the condition that they do not delay these large jobs. In Aggressive backfilling it is checked that jobs moving ahead in the queue do not delay the first queued job, while in Conservative backfilling it is ensured that this job moving does not delay any previous job in the queue [73].

We build the realistic scenario of a grid by simulating a real grid in the Netherlands: The Distributed ASCI Supercomputer 3 (DAS3) [102]. This grid consists of 5 clusters whose details are described in Table 6.1. With DAS3, users can submit jobs through either a local scheduler and run the jobs directly on a cluster or through the grid scheduler KOALA [72] and let the grid scheduler decide to which cluster the jobs are allocated. The grid infrastructure in our simulations resembles those in Table 6.1. We implement the Aggressive backfilling policy for local schedulers. At the grid level, submitted jobs are placed in a waiting queue. Whenever there are free processors, the first job in the waiting queue that requests less than or equal to the number of free processors is submitted to a local scheduler by the grid scheduler. As such, for each simulation of a grid scenario, we need 5 background and 1 grid workloads with a total number of ∼378K jobs.

6.4 Experimental Results

This section consists of two parts. At first, we present our performance evaluation for the single cluster scenario. Then, we finalize this section by considering the performance impacts of job arrivals in a grid.

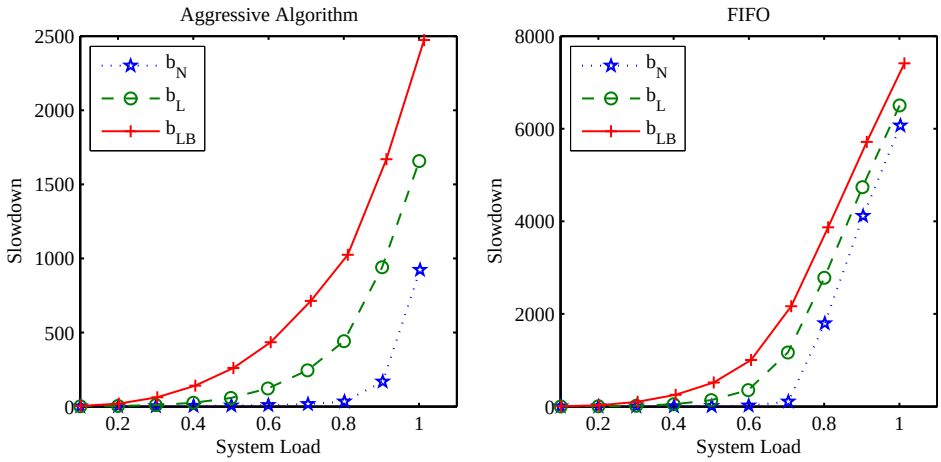


FIGURE 6.1: Performance impacts of LRD and burstiness under two scheduling policies.

6.4.1 Cluster Scenario

This section presents two experiments to answer two questions:

1. What are the impacts of dependence and burstiness in job arrivals on the scheduling performance of a single cluster?

For this experiment, we use the Poisson, MWM and MMWM models to generate 50 workloads b_N, 50 workloads b_L and 50 workloads b_{LB}, respectively. These workloads are then used for simulation of a cluster with GridSim. Figure 6.1 shows the performance impacts of LRD and burstiness under two scheduling policies FIFO and Aggressive. It can be seen that the results are completely consistent in both cases. Without dependence and burstiness, b_N gives the best performance (smallest slowdown). Compared to b_N, b_L degrades the scheduling performance of a cluster since job arrivals of b_L are long range dependent. We argue that this result is caused by the occurence of several high-load periods in b_L. At a high-load period, several jobs are submitted to the cluster. When such a period starts by the occurence of a large arrival rate value, a long sequence of similar values will occur in a rate process with LRD. Therefore, LRD will result in a long queue of waiting jobs and lead to long waiting times. Consequently, LRD causes a performance degradation of the cluster. Finally, b_{LB} achieves the worst performance because it exhibits burstiness in addition to dependence. A high-load period of a bursty workload is often more serious since more jobs arrive than in a non-bursty workload. Consequently with LRD, the bursty workload will contain several serious and long durations of high-load. This causes the cluster to undergo overloaded periods and leads to a severe performance degradation of the system. As such, via this experiment, we conclude that LRD and burstiness

separately or in combination have an adverse impact on the scheduling performance of a cluster.

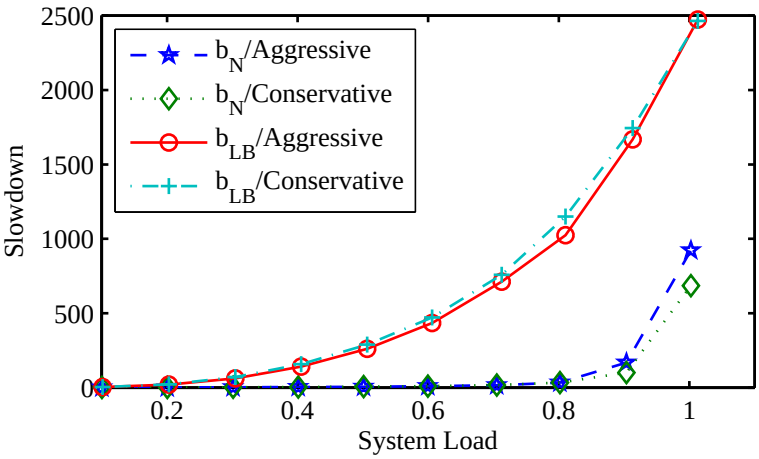


FIGURE 6.2: Compare the scheduling performance of two backfilling policies under two different circumstances.

2. How do long range dependence and burstiness affect backfilling policies of scheduling?

To answer this question, we use 50 workloads b_N and 50 workloads b_{LB} for simulation with GridSim. The scheduling performance of two backfilling policies Aggressive and Conservative is compared under two circumstances: using workloads with (b_{LB}) and without (b_N) dependence and burstiness. As we can see from Figure 6.2, the results differ in two cases. For b_N, both scheduling policies produce similar performance when the system load is lower than 0.8. But Conservative seems to be better than Aggressive if the system load becomes larger than 0.8. In contrast, the result for b_{LB} indicates that Aggressive tends to produce slightly better performance than Conservative. We explain these observations as follows. Being generated by the Poisson model, jobs of b_N arrive regularly since their interarrival times are approximately equal. Therefore, b_N does not have high-load periods. On the contrary, b_{LB} contains several serious and long periods of high-load because of the dependence and the burstiness properties. The occurence of a high-load period causes the cluster to be overloaded and floods the waiting queue of the system. This is an ideal situation for backfilling policies which allow subsequent jobs to leapfrog other jobs in the waiting queue for execution. Compared to Conservative which only backfills a job if it does not delay any previous jobs in the waiting queue, Aggressive is able to do more job backfilling, since it only needs to ensure the backfilled job not to delay the first job in the waiting queue. Therefore, with the existence of several high-load periods in dependent and bursty workloads, it is reasonable that Aggressive schedules jobs better than Conservative. This result is also supported in [73] where Aggressive is shown to

produce better scheduling performance than Conservative in 4 over 5 real traces (for the fifth trace, both policies yield the same performance). As such, we conclude that using workloads with LRD and with burstiness in scheduling evaluation increases the reliability of the result, because real job arrivals are dependent and bursty. Workloads without these characteristics should not be used since they may lead to undependable results as is shown in our experiment with b_N.

6.4.2 Grid Scenario

In our grid scenario, users can submit jobs either directly to a cluster of a grid, represented by a background workload, or through a grid scheduler, represented by a grid workload. Grid jobs will be assigned to the local scheduler of a cluster by the grid scheduler and they will join background jobs of the cluster. Consequently, the dependence and burstiness structures of job arrivals may be broken if workloads exhibit these characteristics and so their performance impacts are difficult to predict. Experiments in this section answer 4 questions:

1. What are the performance impacts of dependence and burstiness in ***background job arrivals*** *on* ***background clusters****?*

As concluded in the experiments of the cluster scenario, dependence and burstiness in cluster job arrivals cause a scheduling performance degradation for the cluster. This question considers whether the outcome changes in case of the grid scenario. Before this experiment, we expected a change of the outcome based on the belief that the dependence and burstiness structures of cluster job arrivals would be broken when grid jobs join with cluster jobs. However, as we can observe in Figure 6.3(a)[2], the conclusion does not change in all cases: b_N with no property has the best performance, b_L with LRD has a worse performance and b_{LB} with LRD and burstiness has the worst performance. So it can be concluded that dependence and burstiness in cluster job arrivals have a severe impact on the performance of a cluster, either when the cluster operates alone or as part of a grid.

2. What are the performance impacts of dependence and burstiness in ***grid job arrivals*** *on* ***background clusters****?*

To answer this question, we fix the background workloads and vary the grid workloads g_i. Considering the results in Figure 6.3(a), we see that the scheduling performance of the cluster does not change when we fix the background workloads to b_N despite the fact that the grid workloads are long range dependent or not and/or bursty or not. Similar results are observed when the background workloads are fixed to b_L and b_{LB}. Hence, we conclude that the dependence and burstiness structures of grid job arrivals do not affect local schedulers.

[2]This figure only draws the scheduling performance of the cluster of Leiden University. For other clusters, the performance results are quite similar.

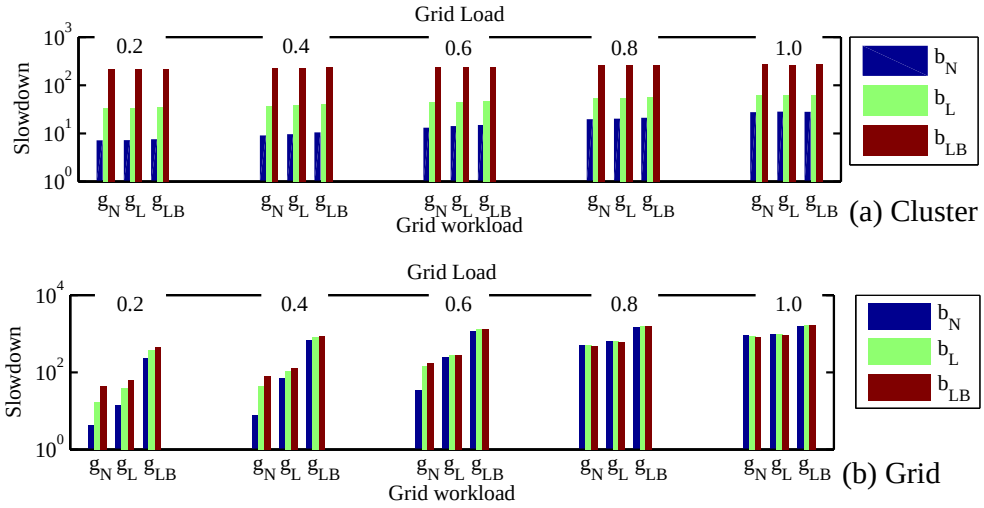


FIGURE 6.3: Performance impacts of background and grid workloads on the cluster of Leiden University (a) and on the grid (b). In the figures, results are divided into 5 groups corresponding to 5 values of grid load from 0.2 to 1.0. Each group contains 3 sub-groups corresponding to 3 grid workload patterns. Each sub-group describes the slowdown of 3 background workload patterns. b_i and g_i denote background and grid workload patterns, respectively.

3. What are the performance impacts of dependence and burstiness in ***background job arrivals*** *on* ***a grid****?*

The answer for this question can be seen in Figure 6.3(b). At first, we fix grid job arrivals to Poisson (g_N) and see that although grid jobs do not affect local scheduling as in the second question, cluster jobs indeed affect grid scheduling. With a low grid load (0.2, 0.4, 0.6), b_{LB} has the worst performance impact on grid scheduling, then b_L and finally b_N. When the grid load becomes higher or equal to 0.8, the performance impact of cluster jobs seems to disappear. Similar observations are drawn from Figure 6.3(b) when we fix grid job arrivals to only LRD or to both LRD and burstiness. However, the grid load, where the performance impact of cluster jobs starts to disappear, decreases to 0.6 for g_L and 0.4 for g_{LB}. Therefore, it can be concluded that LRD and burstiness of background job arrivals may decrease grid scheduling performance but adding these features for grid job arrivals can help to alleviate the negative impact of cluster job arrivals on grid scheduling.

4. What are the performance impacts of dependence and burstiness in ***grid job arrivals*** *on* ***a grid****?*

Before doing this experiment, we thought that the dependence and burstiness structures of grid jobs may not have any effect once the jobs are allocated to different local schedulers and have to wait for execution in different waiting queues. Hence, we

anticipated that these workload features will not have much effect on grid scheduling. However, observing the results from Figure 6.3(b), we see that grid jobs actually perform worse if they are long range dependent and bursty in all cases. As such, although dependence and burstiness of grid jobs can help to reduce the adverse impact of background jobs on the grid scheduler, the inverse direction is not assured.

6.5 Related Work

The topic on scheduling design has been grown since numerous decades [77] and a lot of studies, for example [3, 16, 32, 82, 96, 112, 115], on designing and evaluating scheduling algorithms are based on random workloads that are unrealistic, possibly because there are only a few statistical realistic workload models available in the liturature. In such studies, jobs are submitted to a system with either the Poisson distribution, strictly in sequence or with fix-interval arrivals. Such simple job submission schemes are far from what is observed in real world data as we have shown in Section 3.2, namely that real job arrivals are long range dependent and bursty in many real workloads. Our question is whether the results of those scheduling design studies are still accurate with real workloads. The answer is not clear because the performance impacts of long range dependence and burstiness of job arrivals on schedulers are not well known since there are just a few studies investigating these issues in the context of clusters and grids. To our best knowledge, there are only two studies dealing with performance issues of job arrivals. In [97], Squillante et al. use the time series model ARIMA to fit the interarrival time distribution. Then, they use job arrivals generated by ARIMA to evaluate the performance of schedulers. They conclude that the performance of different scheduling policies within their model do differ from that obtained under a phase-type interarrival time distribution. In [45], Li et al. show that dependence can affect the performance of a grid. Their work is done under the assumption that all jobs are serial. This assumption is clearly not suitable for clusters. Moreover, they do not take into account the performance impact of the burstiness characteristic. Being different from [45, 97], our study focuses on parallel jobs. Furthermore, instead of simply concentrating on fitting the distribution of job arrivals, we drive our study to two real workload properties, namely long range dependence and burstiness, at the same time. In addition, the interaction between background and grid workloads is also considered. To enable our study, representative workload models are required. Although there are several workload models developed for clusters and grids, not many models are dedicated to job arrivals, especially with LRD and burstiness. This is also a reason why the performance impacts of these characteristics are little known in the literature.

Since LRD is a well-known property in communication networks, it is also worthy to mention some related work from that field. With respect to network traffic, studies in [17, 83] demonstrate that long range dependence has considerable impact on queueing performance. However in the context of variable-bit-rate video traffic,

Ryu et al. [87] show that even in the presence of LRD, long-term correlations do not have signifficant impact on the cell loss rate. Different from [17, 83, 87], our study deals with clusters and grids. Since LRD has different impacts under two different network/video traffics, it is not easy to predict its impact on system traffic. In particular, a grid is far from a queueing system since once its jobs are allocated to different background clusters, scheduling of these jobs is out of the control of the grid scheduler. This makes the impact of LRD unpredictable. Moreover, burstiness is an important property that may be considered as a negative impact but its real impact on clusters and grids has not been demonstrated yet.

6.6 Summary

The common presence of LRD and burstiness indicates that it is essential to know their performance impacts on clusters and grids. Our study used model-based simulations to establish real cluster and grid scenarios and explore the performance impacts of these features. In the scenario of a single cluster, our simulation results showed that LRD and burstiness have severe performance impacts and can significantly affect the design of scheduling algorithms, i.e. inaccurate conclusions can be drawn. In the scenario of a grid, we showed that in particular situations, grid and background jobs may join together to improve the scheduling performance. However, in most cases, grid and background jobs do not really affect each other. Future research should include how to improve the scheduling performance of clusters and grids by considering the dependence and burstiness structures of real job arrivals. From our experiments, a mixture of grid and cluster jobs may help to improve grid scheduling in particular situations and that fact should be utilized.

Chapter 7

Performance Impact of Runtime and Parallelism on Scheduling

As we have shown in Section 3.3, temporal locality and the cross-correlation between runtime and parallelism processes exist commonly in a large number of real parallel workloads. Therefore, we argue that it is interesting to discover their performance impacts on scheduling, which will be investigated in this chapter. To our knowledge, there is only one study [57] that takes into account performance issues of correlation[1]. In [57], Lo et al. propose a number of scheduling algorithms which are evaluated under three degrees of correlation. They fix the correlation to -1, 0, $+1$ and show that when the correlation switches from -1 and 0 to $+1$, a good algorithm can turn to be worse than others. The limitation of their study is that the correlations of -1 and $+1$ are unrealistic because real workloads cannot exhibit such perfect correlations as shown in Section 3.3. Being different from [57], instead of only comparing algorithms, we focus our study on showing how the performance of schedulers reacts, i.e., increases/decreases, when changing correlation degrees. In our study, we propose an efficient approach to exactly adjust a correlation degree on demand and thus the performance impact of correlation can be sufficiently evaluated at any desired degree. With respect to performance issues of locality, we are not aware of any study on this, probably due to a lack of available statistical workload models that are able to capture this phenomenon. Feitelson [18] recently indicated that locality is a newly identified phenomenon in parallel workloads and it is essential to investigate its effects on parallel systems.

7.1 Control Correlation

In order to completely evaluate the performance impact of the correlation between a runtime process and a parallelism process on parallel systems, an efficient approach

[1] We use the words "locality" and "correlation" to indicate the phenomenon of temporal locality and the correlation between runtime and parallelism, respectively, throughout this chapter for brevity.

that is able to control its degree on demand is required. We propose in this section such an approach with the idea illustrated in Figure 7.1. Each row of circles/squares in the figure represents one stochastic process that can be a runtime or parallelism process. Assume the two processes in Figure 7.1 exhibit a correlation, we decrease this correlation in two steps. Firstly, we select randomly a number of elements from one process, and secondly permute them. Although there exists a correlation between the selected elements and their partners[2] before the permutation, this correlation will be lost after they are randomly permuted. Consequently, the total correlation between the two processes decreases.

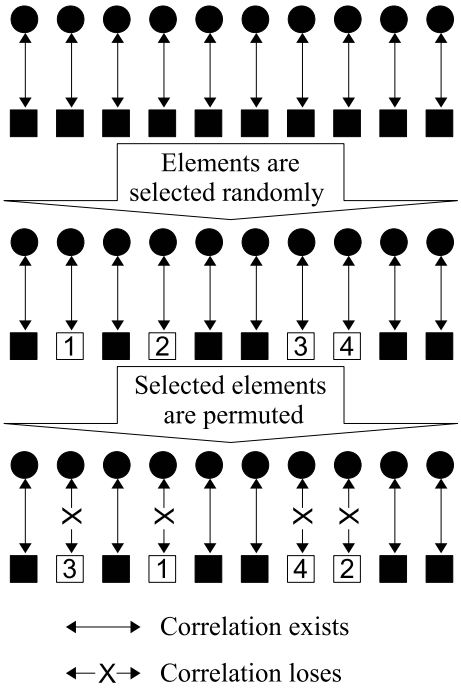


FIGURE 7.1: Illustration of controlling correlation.

Our approach is implemented by a procedure, named *Control_Correlation*, presented in Algorithm 3. The procedure receives a runtime process $\{R_i\}$, a parallelism process $\{P_i\}$ and a list of m parameters $d_j, j = 1, \ldots, m$ as its inputs. The value of d_j ranges from -1 to $+1$, where $|d_j|$ indicates which percentage of elements is randomly selected and kept unchanged (i.e., not permuted) and the sign of d_j indicates a positive/negative correlation, respectively. For example, if the two processes

[2]In Figure 7.1, each element in one process will associate with one element of the other process, showing by either a continuous or discrete two-direction arrow. We use the words "partner" and "partnership" to indicate the associated element and the association between two elements, respectively.

in Figure 7.1 exhibit a negative correlation, we have $d_j = -0.6$ because 60% elements are not permuted. The outputs of the procedure consist of m new parallelism processes $\{P_i^j\}, j = 1, \ldots, m$. Each $\{P_i^j\}$ associates with $\{R_i\}$ to yield a correlation that corresponds to d_j.

Algorithm 3 Procedure *Control_Correlation*.

Input: a runtime process $\{R_i\}$, a parallelism process $\{P_i\}$ and a list of m parameters $d_j, j = 1, \ldots, m$.
Output: m new parallelism processes $\{P_i^j\}, j = 1, \ldots, m$.

// Maximize the correlation
$\{R'_i\} = sort(\{R_i\},' ascend')$;
$\{Pp_i\} = sort(\{P_i\},' ascend')$;
$\{Pn_i\} = sort(\{P_i\},' descend')$;

// To assure that locality is not affected
Rearrange $\{R'_i\}$ to $\{R_i\}$ while fixing its partnerships with $\{Pp_i\}$ and $\{Pn_i\}$;

// Generate m parallelism processes and tune correlation
for $j = 1$ to m **do**
  if $d_j \geq 0$ **then**
    $\{P_i^j\}$ = PERMUTATION($\{Pp_i\}$,$1 - d_j$);
  else
    $\{P_i^j\}$ = PERMUTATION($\{Pn_i\}$,$1 + d_j$);
  end if
end for

function $\{Y_i\}$ = PERMUTATION($\{X_i\}, z$)
 // Calculate number of elements in $\{X_i\}$ *to be permuted*
  $N = round(z \times length(\{X_i\}))$;
  Randomly select N elements in $\{X_i\}$ and permute them
  to form a new process as shown in Figure 7.1;
  Assign the new process to $\{Y_i\}$;
end

The procedure tunes a correlation via 3 steps. Since our approach can only reduce the degree of correlation, the procedure as the first step needs to maximize it to achieve a maximal positive/negative correlation[3]. As presented in Section 2.3.2, we

[3]In theory when a correlation reaches the maximal threshold, the correlation coefficient reaches ± 1. The Spearman's rank correlation coefficient can only obtain ± 1 in case there are no tied ranks, i.e., elements of a stochastic process must have distinguish values so that they do not have the same ranks [60, 75]. However, tied ranks exist in real runtime and parallelism processes due to repeated data values. Therefore, the maximal correlation that can be achieved in practice is often only approximately equal to $+/-1$, which is refered as a maximal positive/negative correlation, respectively.

use Spearman's approach which is based on data ranks instead of the data itself to calculate a correlation coefficient. Hence to maximize the correlation coefficient, we sort $\{R_i\}$ in an ascending order to form $\{R'_i\}$ and sort $\{P_i\}$ in ascending and descending orders to form $\{Pp_i\}$ and $\{Pn_i\}$, respectively. Consequently, we achieve a maximal positive correlation between $\{R'_i\}$ and $\{Pp_i\}$, and a maximal negative correlation between $\{R'_i\}$ and $\{Pn_i\}$. As the second step, we rearrange $\{R'_i\}$ to $\{R_i\}$ since our purpose is to keep the input runtime process unchanged to ensure the procedure will not affect locality. In this rearrangement, we keep the partnerships between elements of $\{R'_i\}$ and elements of $\{Pp_i\}$ and $\{Pn_i\}$ unchanged, i.e., whenever we swap 2 elements of $\{R'_i\}$, we also swap their partners in $\{Pp_i\}$ and $\{Pn_i\}$. After the rearrangement, $\{R'_i\}$ turns back to $\{R_i\}$ which now has maximal positive and negative correlations with $\{Pp_i\}$ and $\{Pn_i\}$, respectively.

As the last step after maximizing the correlation, we control its degree by calling the function PERMUTATION. This function works as explained in Figure 7.1, which is described in the first paragraph of this section. The procedure can be used to create several correlations with different degrees on demand by specifying a list of parameters $\{d_j\}$. The advantage of the procedure is that it yields many $\{P_i^j\}$ but does not affect $\{R_i\}$. This means that though the degree of correlation changes, the degree of locality is kept unchanged. Thus, when evaluating the performance impact of correlation, we can avoid its potential interdependent impact with locality on results.

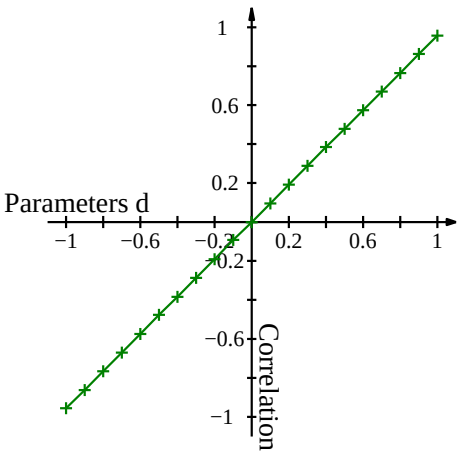


FIGURE 7.2: The relationship between correlation and d_j, shown for the trace HPC.

The last problem we address in this section is the meaning of the list of parameters $\{d_j\}$. As explained above, $|d_j|$ indicates which percentage of elements in a stochastic process is kept unchanged by the PERMUTATION function, and the sign of d_j indicates a positive/negative correlation. However, we would like to know whether d_j has any relation with the degree of correlation. To check this, we do an experiment by applying the procedure *Control_Correlation* on the runtime and the parallelism pro-

cesses of the trace HPC with the list of parameters $\{d_j\} = \{-1, -0.9, -0.8, \ldots, +0.8, +0.9, +1\}$. Interestingly, the result in Figure 7.2, which draws the Spearman's correlation as a function of d_j, shows that there exists a linear relationship between d_j and the degree of correlation. When fitting the line in Figure 7.2, we find that its slope is approximately equal to the maximal correlation that HPC can obtain via the first step of the procedure $Control\_Correlation$: $|\pm 0.95|$. In other words, the line is of the form $Correlation = |MaxCor| \times d_j$, where $MaxCor$ is the maximal correlation obtained in the first step of the procedure. As such, when the possible maximal correlation of a trace reaches ± 1, we can consider d_j as the degree of correlation. Mathematically, this consequence can be proven as follows. Given two N-element stochastic processes X and Y which exhibit a maximal correlation $MaxCor$. With a parameter $d_j \in [-1, +1]$, we select elements randomly to divide X, Y correspondingly into $[(1 - |d_j|) \times N]$-element processes X_1, Y_1 and $[|d_j| \times N]$-element processes X_2, Y_2. In statistics, X_1, X_2 are considered two samples of the population X, and Y_1, Y_2 are two samples of the population Y. As such, we have[4]

- $\mu_{X_1} \simeq \mu_{X_2} \simeq \mu_X$ and $\mu_{Y_1} \simeq \mu_{Y_2} \simeq \mu_Y$,
- $\sigma_{X_1} \simeq \sigma_{X_2} \simeq \sigma_X$ and $\sigma_{Y_1} \simeq \sigma_{Y_2} \simeq \sigma_Y$,
- $\rho_{X_1,Y_1} \simeq \rho_{X_2,Y_2} \simeq \rho_{X,Y} = MaxCor$,

where μ, σ and ρ are the mean, the standard deviation and the correlation coefficient, respectively. Now assume we randomly permute Y_1 as in Figure 7.1, X_1 and Y_1, hence, become uncorrelated and ρ_{X_1,Y_1} reduces from $MaxCor$ to approximately 0. The correlation ρ_{X_2,Y_2} between X_2 and Y_2 is still $MaxCor$ while the correlation between X and Y is reduced to

$$
\begin{aligned}
\rho_{X,Y} &= \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X\sigma_Y} = \frac{\Sigma(X-\mu_X)(Y-\mu_Y)}{N\sigma_X\sigma_Y} \\
&= \frac{\Sigma(X_1-\mu_X)(Y_1-\mu_Y)}{N\sigma_X\sigma_Y} + \frac{\Sigma(X_2-\mu_X)(Y_2-\mu_Y)}{N\sigma_X\sigma_Y} \\
&\simeq \frac{(1-|d_j|)\Sigma(X_1-\mu_{X_1})(Y_1-\mu_{Y_1})}{(1-|d_j|)N\sigma_{X_1}\sigma_{Y_1}} + \frac{|d_j|\Sigma(X_2-\mu_{X_2})(Y_2-\mu_{Y_2})}{|d_j|N\sigma_{X_2}\sigma_{Y_2}} \\
&= \frac{(1-|d_j|)E[(X_1-\mu_{X_1})(Y_1-\mu_{Y_1})]}{\sigma_{X_1}\sigma_{Y_1}} + \frac{|d_j|E[(X_2-\mu_{X_2})(Y_2-\mu_{Y_2})]}{\sigma_{X_2}\sigma_{Y_2}} \\
&= (1-|d_j|)\rho_{X_1,Y_1} + |d_j|\rho_{X_2,Y_2} \simeq |d_j| \times MaxCor = |MaxCor| \times d_j.
\end{aligned}
$$

[4]In statistics, when the size of a sample is large enough, the sample can be used to estimate statistical properties of its population such as mean, standard deviation, etc. In other words, these statistical properties of the sample are approximately equal to those of its population. In our study, each trace contains up to ten thousands of jobs, and hence the statistical properties of the sample are accurate estimators for the properties of the population.

The above proof concludes that our procedure can adjust accurately the degree of correlation on demand by specifying the requested degree via d_j.

7.2 Realistic Simulation Methodology

Similar to the experiments described in Chapter 6, we build our simulation environment on GridSim [5] and use the performance metric *Slowdown* as a function of *System Load*. Applied workload models and scheduling policies are presented in the following subsections.

7.2.1 Applied Workload Model

To form a complete parallel workload for an open-system simulation, we need to generate three important workload attributes: the arrival time indicating the submission time of a job, the runtime indicating how long a job will execute, and the parallelism indicating how many processors a job requests. Since the main theme of our study is about the locality and the correlation characteristics that are present in the runtime and the parallelism attributes, a representative workload model for runtimes and parallelisms with these characteristics is required. Such a model is proposed in [67] and is used in our study. The model receives a runtime process and a parallelism process from a real workload as its inputs. Then it generates a synthetic runtime process and a synthetic parallelism process at random, but preserving correlation and locality of the real workload. In addition, the marginal distribution of the synthetic workload also fits that of the real workload well. In our experiments, we generate job arrivals with the Poisson distribution to ensure that no potential feature of job arrivals affects our evaluation results because some features of job arrivals such as long range dependence and temporal burstiness have significant impacts on parallel system performance as shown in Chapter 6.

In our study, each data point in the performance results, shown by graphs in later sections, is calculated as the average of 20 simulation runs with 20 different workloads. Each workload in our experiments contains 70K jobs, which satisfies the size requirement for a reliable simulation of a scheduler [25]. We generate 340 workloads in total as explained later in Section 7.3, so we simulated ~24 million jobs in our experiments.

7.2.2 Scheduling Policies

We select two scheduling policies in our study, namely first-in first-out (FIFO) and backfilling as they are the most common scheduling policies used in real parallel systems nowadays. Backfilling is a good policy because it is an optimization in the

framework of variable partitioning [73]. The main problem with backfilling is that it requires estimates of job runtimes to be available as accurately as possible. To create a perfect condition for backfilling, we use actual runtimes as the estimates for jobs. Since there are several scheduling algorithms developed with backfilling policy, we choose four common backfilling algorithms, namely Aggressive [73], Conservative [73] and two variations of Selective [98]. With Aggressive, backfilling is done subject to checking that jobs moving ahead in the queue do not delay the first queued job, while with Conservative backfilling it is ensured that this job moving does not delay any previous job in the queue. The main idea of Selective is that reservations are provided selectively only to jobs that have waited long enough in the queue. Selective will classify jobs into different groups. Each group is assigned a threshold to check if a job has waited long enough to get a reservation. In our study, the first variation, the so-called Uncategorized Selective, will assign the same threshold for all jobs. For the second variation, the so-called Categorized Selective, a scheduler will classify jobs into three categories: small (job runtimes are smaller than 1,000 seconds), medium (job runtimes are from 1,000 to 10,000 seconds), and large (job runtimes are larger than 10,000 seconds). When a job arrives at the system, the Categorized Selective scheduler will check to which category it belongs and will use the threshold of that particular category to verify whether a reservation for the job can be granted. As such, we have a total of 5 scheduling algorithms in our experiments.

7.3 Experimental Results

We do two experiments in our studies. The first experiment is to evaluate the performance impact of correlation without the presence of locality. The second experiment considers the effect of locality on scheduling performance under three different circumstances, i.e., with negative, zero and positive correlations.

7.3.1 Performance Impact of Correlation

For this experiment, we create 11 patterns of workload, each corresponds to 11 different degrees of correlation and denoted by $W_j, j = 1, \ldots, 11$. At first, we apply the model in [67] on the trace HPC to generate a synthetic runtime process and a synthetic parallelism process. Since we do not need the presence of locality in this experiment, we randomly shuffle the synthetic runtime process so that it loses the locality structure. Then, we call the procedure *Control_Correlation* with the 2 synthetic processes and a list of parameters $\{d_j\} = \{-1, -0.8, -0.6, -0.4, -0.2, 0, +0.2, +0.4, +0.6, +0.8, +1\}$ as inputs to achieve 11 new synthetic parallelism processes, which associate with the shuffled runtime process to obtain 11 couples of runtime and parallelism processes with correlation ranging from -1 to $+1$. From these couples, we form 11 complete synthetic workloads $W_j, j = 1, \ldots, 11$ by using the Poisson distribution to

generate 11 job arrival processes that are aggregated with the couples. The Poisson parameter λ is calculated such that all workloads produce the same load on the same system. Note that, all 11 workloads have the same runtime process and, therefore, the locality property will not affect their performance. We repeat this workload generation 20 times, so for each degree of correlation we have 20 different synthetic workloads, which results in 220 workloads used in this experiment.

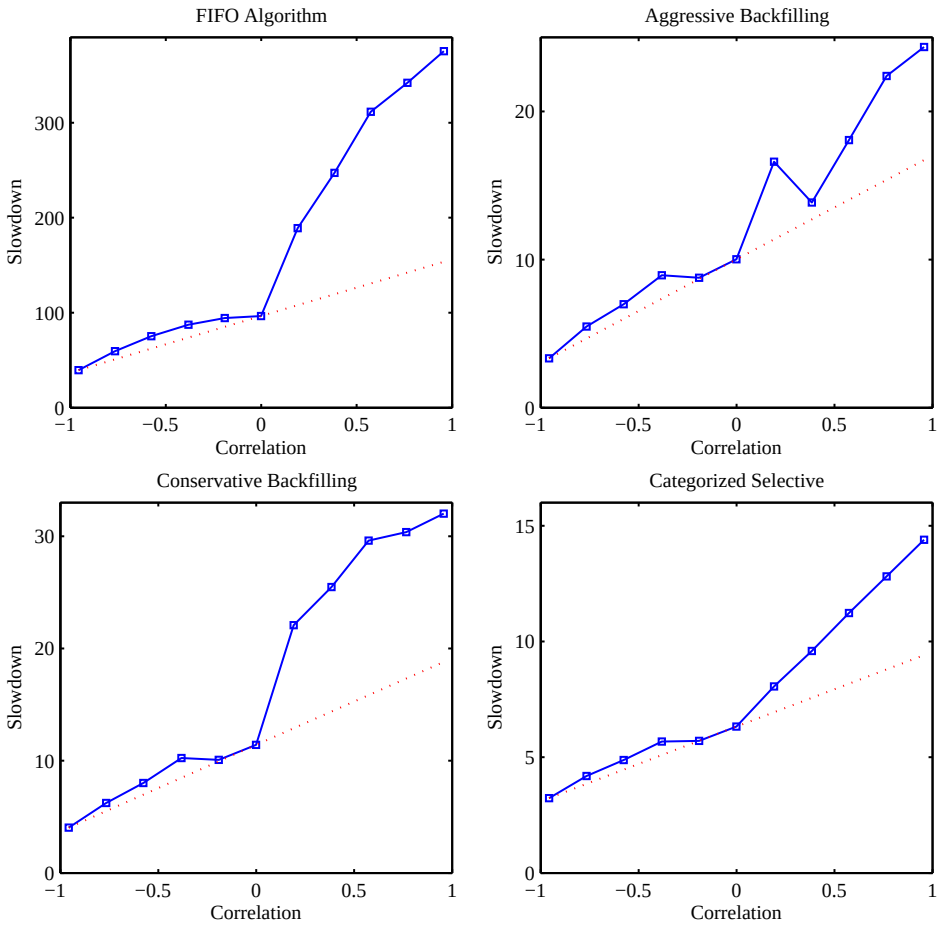


FIGURE 7.3: Performance impact of different degrees of correlation on parallel systems, shown by a solid line. A dot line illustrates the increase tendency of slowdown.

Table 7.1 describes the correlations of the generated synthetic workloads, calculated by Spearman's approach. As we can observe, the results demonstrate that the procedure *Control_Correlation* works well since the degree of correlation can be efficiently controlled as shown in Section 7.1. Another noticable point is that we only change the structure of runtimes (locality) and parallelisms (correlation) but not their

values. Therefore, all the synthetic workloads are representative with respect to the marginal distributions of runtimes and parallelisms since the applied model [67] can fit the marginal distributions well.

TABLE 7.1: Spearman's correlation of synthetic parallel workloads generated with the procedure *Control_Correlation*, presented as *mean* ± *standard deviation* of 20 workloads for each pattern.

Workload	W_1	W_2	W_3
Correlation	-0.957 ± 0.001	-0.766 ± 0.001	-0.576 ± 0.003
W_4	W_5	W_6	W_7
-0.382 ± 0.002	-0.191 ± 0.004	-0.001 ± 0.004	$+0.191 \pm 0.003$
W_8	W_9	W_{10}	W_{11}
$+0.384 \pm 0.003$	$+0.574 \pm 0.002$	$+0.766 \pm 0.002$	$+0.957 \pm 0.001$

All of 220 synthetic workloads are run on the same simulated parallel system with 240 processors (the real system size of the cluster HPC). Figure 7.3 shows the performance impact of correlation on schedulers. We can see that for all cases, a larger degree of correlation leads to a performance degradation (larger slowdown). The reason is that when the correlation increases, longer runtimes tend to associate with larger numbers of processors and therefore, short jobs have to wait longer once processors are allocated to long jobs and this results in poorer performance. Especially, it can be seen in Figure 7.3 that the scheduling performance decreases slowly when the correlation changes from -1 to 0 but then rapidly decreases when the correlation becomes positive and reaches $+1$. The reason of this difference lies in the fragmentation problem. With a negative correlation, the number of processors associated with a long job is not so large and hence, a short job has more chance to be allocated with enough processors for execution. In general, the fragmentation issue happens rarely. However, with a highly positive correlation, the fragmentation problem seems to occur regularly once a long job is in execution and occupies a large number of processors. In this case, the number of free processors is small and not enough for waiting jobs. This leads to a fragmentation and a low utilization of the system. As a consequence, when a positive correlation reaches $+1$, the fragmentation issue becomes more serious and the scheduling performance rapidly decreases. Another notable point drawn from Figure 7.3 is that the fragmentation issue of Aggressive and Categorized Selective policies is considerably alleviated compared with that of FIFO and Conservative policies. We explain this result by the backfilling behaviour. Fragmentation occurs when free processors are not allocated to waiting jobs due to reasons like the number of free processors is not enough or it is enough for a waiting job but the job cannot leapfrog other jobs that are waiting before it in a queue. This situation happens with FIFO policy, making the fragmentation issue worse. It also happens

with Conservative because despite backfilling is supported, this policy only backfills a job if it does not delay any previous jobs in the waiting queue. On the contrary, Aggressive and Categorized Selective are able to do more job backfilling, hence reduce the fragmentation issue. Therefore, we conclude that backfilling is especially good for positive correlation workloads. Since real workloads exhibit correlations ranging from around -0.3 to $+0.5$, we recommend that this correlation range should be carefully considered in evaluating scheduling algorithms.

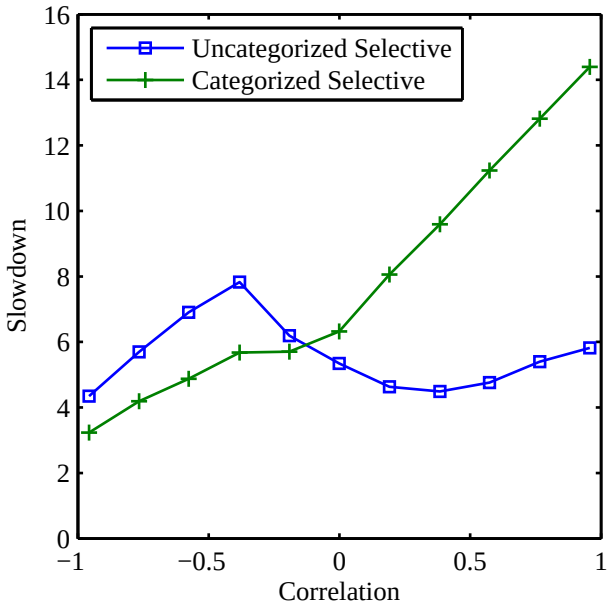


FIGURE 7.4: Compare the performance of two scheduling algorithms under the impact of correlation.

Next to the performance impact of correlation, we also investigate how this feature affects scheduling design. Figure 7.4 compares the scheduling performance of two scheduling algorithms, namely Categorized Selective and Uncategorized Selective, under different degrees of correlation. The result is interesting since Categorized Selective is better than Uncategorized Selective with a negative correlation but turns to be worse when the correlation becomes positive. The reason is that Uncategorized Selective uses the same threshold for all jobs so that if a job has waited long enough, it will be provided with a reservation. Therefore, despite of the correlation, all jobs only have to wait up to the threshold to get a reservation for execution. This explains the relatively similar slowdowns for Uncategorized Selective when the correlation ranges from -1 to $+1$. In contrast, Categorized Selective uses different thresholds for different categories of jobs. Hence, one category can affect the wait times of the other categories. This results in an increase of the slowdown when the correlation reaches $+1$. Consequently, Categorized Selective turns to be worse than

Uncategorized Selective with an increase of correlation though it can be better with a negative correlation. Definitely, there is a potential risk of inaccurate scheduling design if using workloads without or with wrongly modeled correlation.

7.3.2 Performance Impact of Locality

For this experiment, we fix the degree of correlation and enable/disable the locality feature to see how locality affects the performance of schedulers. Correlation is controlled at 3 degrees: -0.2, 0 and $+0.4$ since real workloads exhibit these correlations as shown in Table 3.4. For negative correlation workloads, we apply the model in [67] on the trace FS3 20 times to obtain 20 workloads whose correlation and locality are similar as in FS3. We refer to this kind of workload as W_L. Then we process these workloads to achieve 20 other workloads without changing the correlation but distorting the locality. This kind of workload is denoted as W_N. To disable the locality feature of a workload while keeping its correlation, we randomly shuffle the runtimes and the parallelisms of the workload at the same time, i.e., when we swap the runtimes of 2 jobs, we also swap their numbers of processors. Similarly, we obtain 20 workloads W_L and 20 workloads W_N with zero correlation by applying the model on FS0. For positive correlation workloads, we use FS2 as the input of the model. As such, we have to simulate 120 synthetic workloads with 8,400,000 jobs in this experiment.

Figure 7.5 shows the performance impact of locality on scheduling. We can see that the results are consistent despite the correlation. With regard to Aggressive policy, locality causes a performance degradation for a parallel system. Similar effects are observed for FIFO policy but only if the system is underloaded ($\textit{System Load} < 1$). On the contrary when the system is overloaded, locality indeed helps improve the scheduling performance of FIFO. The reason lies in short jobs. When a system is overloaded, any submitted job is hard to be executed immediately. Instead, it has to wait some time for free processors. With locality, short jobs tend to arrive after and wait for other short jobs. Hence, the wait time of short jobs is often small. In contrast, without locality, short jobs tend to arrive after long jobs. This makes the wait time of short jobs larger and decreases the scheduling performance. However, this situation would not happen with backfilling policies because a short job is able to leapfrog waiting long jobs for execution. As such, locality is harmful for backfilling policies but can be useful for non-backfilling algorithms in case a system is overloaded.

7.4 Discussions

Our experimental results in Section 7.3 provide several clues that can help to design efficient scheduling policies. For illustration, we describe briefly in this section our idea to exploit the correlation and the locality features for enhancing scheduling performance. We define 4 states for a parallel system, consisting of overload-positive,

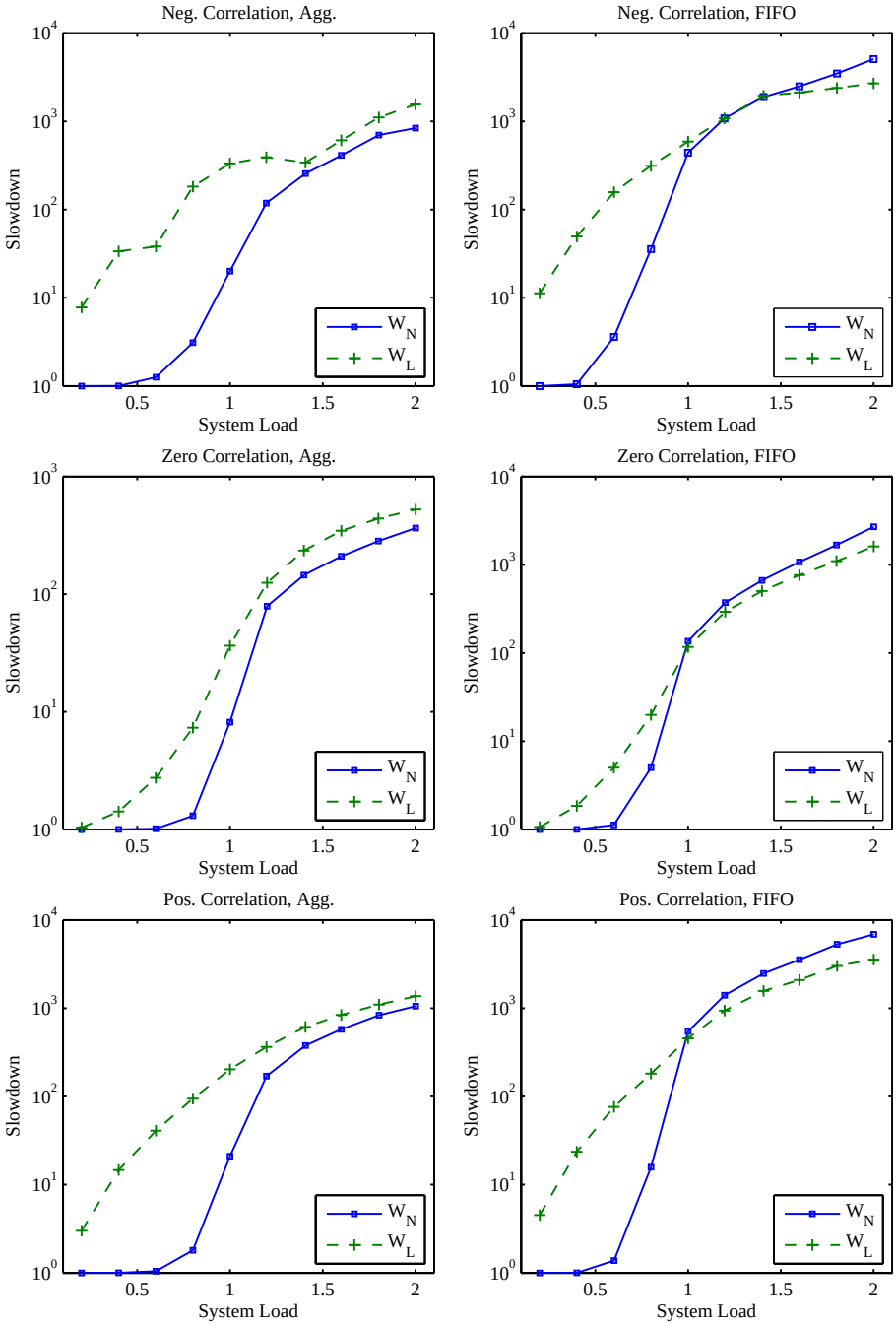


FIGURE 7.5: Performance impact of locality on parallel systems.

overload-negative, underload-positive and underload-negative. The states are mostly self-explanatory by their names, i.e., each represents the status of the system: whether the system is over/under-loaded and whether the workload running on the system exhibits a positive/negative correlation. Then we develop an individual scheduling policy for each state. For example, a backfilling policy would be a good choice for positive correlation workloads to alleviate the fragmentation issue as explained in Section 7.3.1. In particular, a backfilling scheduling algorithm that restrains the wait time of a job to a certain threshold like Uncategorized Selective is relatively good for a positive correlation as shown in Figure 7.4. In addition, a non-backfilling algorithm that well utilizes the locality feature can be used when the system is overloaded. Whenever the system falls into a state, the corresponding scheduling policy of the state is called. For our approach to work, it is important to predict the future state of the system. It is easy to know if a system is under/over-loaded by considering the waiting queue of the system and the status of computing nodes. An efficient predictor needs to be developed for anticipating a state with negative/positive correlation. We tried a relatively simple predictor, that is to estimate the correlation of one week by the average of that over the previous 2 weeks. We divide each long-term trace in Table 3.4 into a series of consecutive weeks and evaluate this predictor. For a week, we compare the estimated correlation with the real correlation of the week. If both give the same positive/negative correlation, we count the week as one correctly predicted time. Consequently, the accurateness in term of number of times that are correctly predicted, is approximately 74% on average in the 12 traces in Table 3.4. Note that, this is just a simple predictor and an advanced predictor should be developed. As a whole, it is predictable for the future state of the system and, therefore, our idea is feasible giving us a promising future work.

7.5 Summary

This chapter studied the performance impacts of two parallel workload features, namely locality and correlation, on scheduling by means of simulation. Our experimental results showed that a positive correlation has more serious impact on parallel system performance than a negative correlation but backfilling policies can help to reduce the impact. Furthermore, scheduling design results may differ significantly under different degrees of correlation. With respect to locality, it decreases the scheduling performance of a parallel system but when the system is overloaded, the feature turns to be useful for non-backfilling policies. Therefore, we conclude that any efficient scheduling design should take correlation and locality into account for a good evaluation. To enable such designs, we proposed in this chapter a procedure, namely *Control_Correlation*, that can help to tune the degree of correlation. The efficiency of the procedure was explained and proved in Section 7.1 as well as practically evaluated in Section 7.3.1. Future research includes developing an efficient scheduling mechanism for parallel systems that takes into account the correlation and the locality features of practical workloads.

Chapter 8

A History-Based Predictor of Parallel Application Runtimes

Backfilling algorithms [73] have become more popular for scheduling in many space-shared computing resources. For instance, the clusters in the High Performance Computing Center North in Sweden [34] and the National Center for Supercomputing Applications in USA [78] all use Maui/Moab [61] scheduler which enables backfilling. Also the SDSC Blue Horizon supercomputer [88] implements backfilling policies, etc. Recall that backfilling means that short jobs which are expected to terminate in time are allowed to leapfrog and execute before previously queued large jobs provided that they do not delay these large jobs [73]. With backfilling, an estimate of the runtime is necessary for each job to determine whether it is short enough to be backfilled. The estimate can be either provided by the user or predicted automatically by the system.

Most parallel systems that implement backfilling policies ask users for the estimate of the runtime with an assumption that users would provide accurate estimates since their jobs could start faster if the estimates are tight but would be killed in case the estimates are too small. However, the user requested runtime[1] is found to be extremely inaccurate because users are motivated to overestimate their jobs so that jobs will not be killed much stronger than to provide accurate estimate to enable jobs to start faster, according to the study in [73]. Although studies in [73, 20, 93, 113, 114] show that accurate requested runtimes only have a small impact on the performance of FCFS-backfilling systems, Chiang et al. have indicated that this result is correct on FCFS-backfilling systems, but not on systems using high performance backfilling policies such as L$\sqrt{X}$F&W-backfill, S$\sqrt{T}$F&W-backfill and LXF&W-backfill [9, 10]. It means a more accurate estimated runtime can improve system performance significantly [9]. Therefore, user estimated runtime should not be used as a runtime predictor in backfilling systems if possible.

With respect to the system predictor, several studies have shown that using histori-

[1]The phrases "user requested runtime" and "user estimated runtime" are used interchangably with the same meaning throughout this chapter.

cal data for runtime prediction can considerably improve the accuracy of the estimates [14, 104, 89, 29, 92]. Nevertheless, these studies were done in a context that does not take care of the problem of underestimation. In theory with backfilling systems, jobs that are underestimated will be killed when they exceed their declared runtime [73]. However, user jobs cannot be killed just because they are underestimated by a system predictor. In this case, the system can only let underestimated jobs continue to run and re-schedule jobs in the waiting queue. Consequently, more jobs may be backfilled and several jobs in the waiting queue will be delayed and finally it leads to an adverse impact on system performance.

Both inaccuracy and underestimation in prediction are not good for backfilling systems. Using user estimated runtime can avoid the problem of underestimation but it is inaccurate. System predictor using recent studies [104, 50, 55, 94] can obtain more accuracy but the problem of underestimation is not solved. Therefore in this chapter, we present a novel predictor which tries to reduce the number of jobs that are underestimated and reduce the prediction error as much as possible.

8.1 Related Work

It is well known that workloads on parallel systems are highly repetitive, because users tend to run the same applications over and over again [15, 21]. This means that the estimate of job runtime can be done based on the information of jobs that finished in the past. In fact, several studies have been proposed to estimate job runtime using historical information. Basically, we can divide these studies into two kinds based on the way they look for similar jobs: categorization and instance based learning.

Predicting studies with categorization consist of [13, 104, 29, 92]. Each of them applies a template of job characteristics to determine similar jobs that already finished in the past. In [13], the system queue is used to categorize jobs into classes and a model is created for each class to predict execution times. Gibbons [29] defines a historical application profiler to classify parallel applications in categories based on static templates including attributes such as user, job name and system queue. Parallel applications are grouped according to these templates and the average execution time of each group is used to predict execution times of future applications. Smith et al. [92] find that using static templates to classify jobs is simple and a more sophisticated technique can be applied. They use a genetic algorithm [65] to dynamically determine which job characteristics produce the best definition of similarity. The most recent study [104] finds that using only the previous two jobs submitted by the same user will also give a good prediction. This predictor is really attractive due to its simplicity.

Studies in [50, 89, 94] use the same instance based learning technique to predict parallel application runtimes. With this technique, information about N most recent finished jobs (called the experience base) is kept. The runtime of a future job is

estimated by searching K nearest neighbor jobs (jobs that are most similar to the one being predicted) in the experience base. The similarity between job X and job Y is measured by a distance function

$$D(x,y) = \sqrt{\frac{\sum_{a=1}^{m} w_a \times d_a(x_a, y_a)^2}{\sum_{a=1}^{m} w_a}}, \tag{8.1}$$

where x and y are attribute vectors of X and Y, respectively. In addition, w_a is the weight and d_a is the corresponding distance function for attribute a with

$$d_a(x_a, y_a) = \begin{cases} 0 & \text{if } a \text{ is nominal, } x_a = y_a \\ 1 & \text{if } a \text{ is nominal, } x_a \neq y_a \\ \frac{|x_a - y_a|}{max_a - min_a} & \text{if } a \text{ is numeric scalar.} \end{cases} \tag{8.2}$$

Once a set of K nearest neighbors are identified, an induction model is applied to generate predictions. The induction models used in [50, 89, 94] include Weighted Average (WA) and Linear Locally Weighted Regression (LLWR), where WA is shown to be better than LLWR. Model WA estimates the execution time by the following function

$$Estimate(job) = \frac{\sum_{i=1}^{K} e^{-(\frac{D(job,job_i)}{\sigma})^2} \times R(job_i)}{\sum_{i=1}^{K} e^{-(\frac{D(job,job_i)}{\sigma})^2}}, \tag{8.3}$$

where $R(job_i)$ is the actually runtime of job i in the set of K nearest neighbors and σ is the kernel bandwidth, which can be a fixed constant value or can be set to the largest distance in the nearest neighbor set.

The differences between the studies in [50, 89, 94] are the applied parameters such as the experience base size N, the neighbor size K, the weights w_a in Eq. (8.1) and the kernel bandwidth σ in Eq. (8.3). These parameters are assigned with fixed values by Senger et al. in [89]. More flexibly, Smith et al. [94] and Li et al. [50] use a genetic algorithm [30] to determine the best values for these parameters. An implementation of this technique is available under the name "Performance Data Miner" [81].

8.2 Workloads Under Study

Table 8.1 describes details of the traces used in this study. Note that, these traces are different from those described in Section 3.1. SDSC02 and SDSC04 are two separate traces collected from the San Diego Supercomputer Center Intel Paragon machine,

whose scheduler is Catalina [7]. SDSC04 includes 13-month data and we use the whole trace in our experiments. HPC2N is from a 120-node Linux cluster named Seth at the High Performance Computing Center North in Sweden [34]. Seth is scheduled by Maui [61]. This trace contains three and a half years worth of accounting records, starting from Jul 2002 to Jan 2006. We use the last two years of the trace as two separate traces in our study under the name HPC2N04 and HPC2N05. Note that, in our study, we only take into account jobs that finished successfully, i.e. jobs that have status equal to 1 in the trace. Jobs that have not finished yet should not be used in prediction because their runtimes are not actual. All traces and detailed information are available on [80]. Although a few newer traces can be found on [80] (with the newest one collected till Jun 2007), we do not select them in our experiments because these traces lack the information of user requested runtimes which is important for our predictor. Traces in Table 8.1 are - to our knowledge - the most recent ones that have the information of user estimated runtime.

TABLE 8.1: Workloads used in the experiments.

Trace	Period	Processors	Number of jobs
SDSC02	01/2002-12/2002	1152	80756
SDSC04	03/2004-03/2005	1664	66743
HPC2N04	01/2004-12/2004	240	81113
HPC2N05	01/2005-01/2006	240	55389

8.3 Predict Application Runtime

Our method to predict application runtimes is presented in this section. The assumption that similar jobs will have similar runtimes is applied in our predictor as in studies mentioned in Section 8.1. However, the method we use to define the job similarity and calculate the estimates is quite different. We define the job similarity by a combination of the categorization and the instance based learning methods, with a new similarity function (see Eq. (8.5) in Section 8.3.1). Moreover, when predicting the runtimes, we do take care of the problem of underestimation. We also notice different impacts of small jobs versus big jobs and separate their roles.

8.3.1 Define Job Similarity

Due to the limitation of information contained in traces used in experiments, we can only select and take into account seven job attributes to define job similarity. They are divided into two groups: nominal and numeric attributes. The nominal

group includes "user name" (u), "group name" (g), "queue name" (q) and "job name" (j). The numeric group consists of "requested number of processors" (c), "requested runtime" (r) and "requested memory" (m). Any other potentially useful attributes such as partition number and application arguments should be added to one of these two groups depending on the kind of attributes.

Our idea of defining the job similarity is a combination of the categorization and the instance based learning methods. Nominal attributes are used to categorize jobs and numeric attributes are used in determining the similarity by a new function (see Eq. (8.5)).

Given two jobs:

- $J_1 = (u_1, g_1, q_1, j_1, c_1, r_1, m_1)$
- $J_2 = (u_2, g_2, q_2, j_2, c_2, r_2, m_2)$.

In order to determine whether these two jobs are similar, we first need to find a best template to classify them. The template can be any subset of the set (u,g,q,j) including the empty subset[2]. For example, the template can be (u), (g), (u,j), or (). The question "how to select the best template" is answered in Section 8.3.3. Once the template is determined, these two jobs will be classified. If they fall into the same category, they are considered to be similar. Otherwise, they are not similar. For instance, if the best template found is (u), J_1 and J_2 are similar in case they have the same user name.

Once J_1 and J_2 are determined to be similar, we need to calculate their similarity using numeric attributes. To avoid the phenomenon where an attribute can overpower the other attributes due to its large range, a linear normalization function $f(x)$ is used to reduce numeric attributes in the range $[0, 1]$

$$f(x) = \frac{x - min_x}{max_x - min_x}. \tag{8.4}$$

After normalizing $c_1, c_2, r_1, r_2, m_1, m_2$ by Eq. (8.4), we calculate the similarity between J_1 and J_2 by

$$Sim(J_1, J_2) = \frac{c_1 \times c_2 + r_1 \times r_2 + m_1 \times m_2}{\sqrt{c_1^2 + r_1^2 + m_1^2} \times \sqrt{c_2^2 + r_2^2 + m_2^2}}. \tag{8.5}$$

The similarity is stronger if the value of $Sim(J_1, J_2)$ is large.

[2]An empty subset is symbolized as ()

8.3.2 Predict the Runtime for a Future Job

We present in this section the idea to predict the runtime for a future job J. Firstly, we look for a set of jobs that are most similar to J and then the runtime of J is estimated based on the runtimes of jobs that belong to this set.

Search for a Set of Similar Jobs

First, we need to save N recent finished jobs in the historical database. Once a future job J arrives in the system, we search in this database K, $K < N$, jobs that are most similar to J (these K jobs are called a set of nearest neighbors). The approach to define the similarity between two jobs is described in Section 8.3.1. With this approach, it is possible to find no job similar to J in the saved database using the best found template (see Section 8.3.3 for determining the template). In this case, we simply replace the best found template by the empty template[3]. It means all the jobs in the database are applied by Eq. (8.5) to calculate the similarity with J. The parameters K and N are determined in Section 8.3.3.

Estimate the Runtime

Once a set of K nearest neighbor jobs are identified, we estimate the runtime of the future job J by the average of the runtimes of these K jobs

$$Est(J) = \frac{\sum_{i=1}^{K} R_i}{K}, \tag{8.6}$$

where R_i is the actual runtime of job ith in the set of nearest neighbors. A weighted-average-based estimation similar to Eq. (8.3) can be applied, but we find in our experiments that this idea is not better than the estimation in Eq. (8.6).

A simple way proposed in [73] to reduce the possibility that J is underestimated is to add $1\frac{1}{2}$ times the standard deviation of the runtimes of these K jobs. However, this method can lead to an inaccuracy and that increases the prediction error. Therefore, to reduce the inaccuracy, we will add to the estimate the standard deviation multiplied by a factor α as

$$Est(J) = \frac{\sum_{i=1}^{K} R_i}{K} + \alpha \times std(\{R_i\}), \tag{8.7}$$

[3]Other choices can be applied. For example, we can remove some elements from the best found template to form a new template. However, it is not easy to generally determine elements that should be removed. Therefore, we decide to remove all elements in our study.

where $std(\cdot)$ indicates the standard deviation and the factor α will be determined in Section 8.3.3 in such a way that it decreases the inaccuracy as much as possible.

Adding the standard deviation can reduce the problem of underestimation but can also potentially cause the problem of overestimation. However, we find that the problem of overestimation can be partly limited by using the user requested runtime. This is because in backfilling parallel systems users are motivated to overestimate their jobs so that jobs will not be killed. Therefore, the user requested runtime can be used as a good upper bound for the estimate of the runtime of J. Figure 8.1 gives an illustration about the ratio between the actual runtime and user requested runtime per job. We see clearly that most of the ratios are smaller than 1. This means that even also another variable such as the user requested runtime multiplied by a factor β, $\beta < 1$, can be used as the upper bound. Therefore, we use the user requested runtime multiplied by a factor β as the upper bound in our study as in the following equation

$$Est(J) = min\left(\frac{\sum_{i=1}^{K} R_i}{K} + \alpha \times std(\{R_i\}), \beta \times r_J\right), \tag{8.8}$$

where r_J is the user requested runtime of J and the factor β is determined in Section 8.3.3.

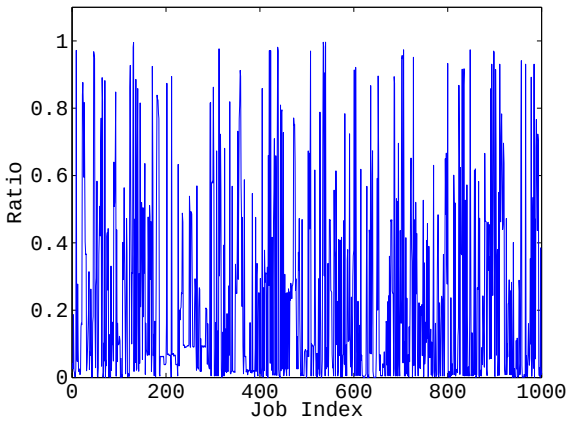


FIGURE 8.1: Ratios between actual runtimes per user requested runtimes, extracted from SDSC04.

8.3.3 Train for Best Parameters

As indicated in Sections 8.3.1 and 8.3.2, there are several parameters that need to be determined. They include the template to categorize jobs, the historical database size N, the number of nearest neighbor jobs K, the factor α and the factor β. However, we found that it is not reasonable if all jobs are applied with the same parameters to predict their runtimes. Instead, jobs should be divided into groups and each group should have it own parameters.

Separate Jobs

The user requested runtime is clearly an upper bound for estimating the runtime of a job. Therefore, we know that the actual runtime of a job is really small if its requested runtime is small. However, no conclusion can be made if the requested runtime is big. By using the user requested runtime as the upper bound in Eq. (8.8), we have reasons to believe that many of the prediction errors belong to jobs that have big user requested runtimes. To check this idea, we implement the most recent predicting method proposed in [104]. This is a categorization-based approach. Its idea is that the runtime estimate for a future job is calculated by the average of the actual runtime of the two most recently finished jobs submited by the same user. We apply this predictor on the trace SDSC04 and manually select 20,000 seconds as pivot to separate jobs[4]. This means that jobs that have a user requested runtime smaller than 20,000 seconds are considered to be small. Otherwise, they are considered as big jobs. Interestingly, the results in Table 8.2 show that although there are only a small number of big jobs (23.6%), they contribute to a considerable prediction error (74%). We explain this situation as follows. If a future job has a big user requested runtime, its actual runtime has a high probability to be big. Therefore, if we want to have an accurate estimate for this future job, we need to find similar big jobs. However, big jobs in fact usually have not terminated yet at the time of making the prediction. Consequently, short jobs that already terminated are often found as similar jobs. This will lead to the situation of underestimation and create a considerable prediction error. In order to solve this problem, we need to look further in the past with a hope that at the time of making the prediction similar big jobs are already completed. However, looking so far in the past is not good for small jobs because it is proven that parallel jobs are highly repetitive [15, 21]. This means that a recency characteristic is necessary for runtime prediction of small jobs. Therefore, our new idea to solve this issue is that we separate big jobs from small jobs and determine individual parameters for each group. The following is a simple example for this issue.

Assume that we have 6 jobs:

[4]The manual selection is just to check our belief that many of the prediction errors belong to jobs that have big user requested runtimes. The possible best value for this pivot parameter will be determined in the training step of our predictor.

TABLE 8.2: Contribution of small jobs compared with big jobs in the absolute prediction error using SDSC04.

	Big jobs	Small jobs
Quantity	23.6%	76.4%
Absolute error	38514 hours (74%)	13584 hours (26%)

- $J_1 = (u1, g1, q, app1, 10, 500, 10, 1, 450)$,
- $J_2 = (u2, g2, q, app2, 4, 110, 2, 3, 100)$,
- $J_3 = (u1, g1, q, app1, 11, 520, 10, 480, 500)$,
- $J_4 = (u2, g2, q, app2, 5, 50, 3, 490, 40)$,
- $J_5 = (u2, g2, q, app2, 4, 90, 2, 550, 30)$,
- $J_6 = (u1, g1, q, app1, 11, 500, 12, 560, 480)$,

where each job is formatted as (user name, group name, queue name, job name, requested number of processors, requested runtime, requested memory, arrival time, actual runtime). At the time J_6 arrives the system (560), if we do not look so far in the past to predict the runtime for J_6 (for example the historical database size $N = 3$ with J_3, J_4, and J_5 are in the database), we will not find any job that is really similar to J_6. Although J_3 is a good candidate, we do not know exactly its actual runtime since at the time we predict the runtime for J_6 (560), J_3 has not terminated yet. It is because J_3 arrives the system at the time 480 and runs 500 seconds, then it will terminate at the time 980. As such, we need to look further in the past by increasing N to find similar jobs for J_6. If $N = 5$, J_1 can be used to yield a good runtime estimation for J_6 since J_1 already terminated. However, the estimation for J_5 will be inaccurate in this case. It is because J_2 is selected as the similar job for J_5 in case $N = 5$ instead of J_4 in case $N = 3$. This is clearly an issue: a big historical database size is better for big jobs while a smaller value of N is better for small jobs due to the recency characteristic.

Train Parameters

We firstly summarize in this section all the parameters that have to be determined and then present the approach to obtain them.

All necessary parameters in our study include:

- A template to classify jobs as indicated in Section 8.3.1. This template is represented by a set $(x_1, x_2, x_3, x_4), x_i \in \{0, 1\}$, where x_1, x_2, x_3, x_4 represent user, group, queue and job name, respectively. If $x_i = 1$, the corresponding job attribute will be inserted to the template (and removed from the template if $x_i = 0$).
- Because we need to separate jobs using the user requested runtime attribute r, a pivot parameter $Pivot$ needs to be determined, where $min(r) < Pivot < max(r)$. It can be flexible to separate jobs in a more detailed way and obtain more accurate prediction. For instance, we can divide jobs into three groups: small, medium and big jobs. In that case, just simply define two pivot parameters $Pivot_1$ and $Pivot_2$.
- For each group of jobs, we need to determine the following parameters: the historical database size N, the number of nearest neighbor jobs K, the factor α and the factor β. We separate jobs into two groups in our study, hence we need $(N_1, K_1, \alpha_1, \beta_1)$ for the first group and $(N_2, K_2, \alpha_2, \beta_2)$ for the second group.

In our study, we divide each trace into two parts: the first part is used for training and the second part is used for testing. To determine the above parameters, we apply a genetic algorithm [30] on the training part and then use these parameters for the testing part to make predictions. A genetic algorithm will evolve individuals throughout a number of generations. For each generation, the algorithm will evaluate individuals in the population, select good individuals, cross and mutate them to produce the next generation. This work is repeated until a stopping condition is satisfied. We use a maximum number of generations as stopping condition.

Our individual representation for the training parameters (described above) is as follow

$(x_1, x_2, x_3, x_4, Pivot, N_1, K_1, \alpha_1, \beta_1, N_2, K_2, \alpha_2, \beta_2)$.

A fitness function is used to evaluate each individual. It is chosen so that good individuals have higher fitness and therefore have higher chance to be selected for producing the next generation. Because our objective is to reduce the number of jobs that are underestimated as well as to decrease the prediction error as much as possible, we select the following fitness function

$$Fitness = \frac{-NormalizedError}{e^{(1-PercentUnderestimate)^2}}, \tag{8.9}$$

where $NormalizedError = \sum_i |P_i - R_i| / \sum_i R_i$, P_i and R_i are the predicted and actual runtime of job ith, respectively. $PercentUnderestimate$ is the percentage of underestimated jobs, $PercentUnderestimate \in [0, 1]$. We use exp in the denominator of the fitness function because we want to reduce the impact of $PercentUnderestimate$ and increase the impact of $NormalizedError$ on the function.

8.4 Experimental Results

The workloads used in our experiments are described in Section 8.2. Training parameters obtained for each workload are shown in Table 8.3. The quality of our predictor is compared with the predictors proposed in the most recent studies [50, 104]. For the categorization-based approachs, we select the predictor proposed by Tsafrir et al. in [104]. It predicts the runtime of a future job by averaging the runtimes of the two most recent terminated jobs submitted by the same user. This simple predictor is demonstrated to significantly improve the prediction accuracy. For the instance-based learning approach, we select the most recent predictor proposed by Li et al. in [50]. An implementation of this predictor is available on [81]. We refer the first predictor as Tsafrir's and the second predictor as Li's.

TABLE 8.3: Training parameters obtained for each trace.

Parameters	SDSC02	SDSC04	HPC2N04	HPC2N05
x_1	1	1	0	1
x_2	0	1	1	0
x_3	1	0	1	0
x_4	1	1	0	0
Pivot	19369	40931	32257	122562
N_1	5046	6268	6904	4763
K_1	10	8	8	9
α_1	0.956	0.976	0.997	1
β_1	0.949	0.958	0.976	0.989
N_2	8661	9757	9189	5610
K_2	5	2	2	10
α_2	0.105	0.038	0.021	0.008
β_2	0.941	0.925	0.986	0.832

8.4.1 Metrics Used in Evaluation

We use two metrics to evaluate the accuracy and one metric to evaluate the problem of underestimation.

Underestimation

We use the percentage of underestimated jobs to evaluate the problem of underestimation since we want to reduce the number of jobs that are underestimated as much as possible.

Mean Absolute Error (MAE)

The mean absolute error of N jobs is calculated by

$$MAE = \frac{\sum_{i=1}^{N} |P_i - R_i|}{N}, \tag{8.10}$$

where P_i and R_i are the predicted and actual runtime of job ith, respectively. Another metric called *normalized error (NE)* can also be used

$$NE = \frac{\sum_{i=1}^{N} |P_i - R_i|}{\sum_{i=1}^{N} R_i}. \tag{8.11}$$

Since we apply the same traces for all predictors, it means $\sum_{i=1}^{N} R_i$ does not change. Hence, we only use the metric mean absolute error in our evaluation.

Weighted Absolute Error (WAE)

Proposed in [108], the metric weighted absolute error is motivated by the fact that a larger and longer job with a prediction error should have more impact on other jobs than a smaller and shorter job with the same prediction error. The weighted absolute error of N jobs is calculated by

$$WAE = \frac{\sum_{i=1}^{N} |P_i - R_i| \times C_i \times R_i}{\sum_{i=1}^{N} C_i \times R_i}, \tag{8.12}$$

where P_i, R_i and C_i are the predicted runtime, actual runtime and requested number of processors of job ith, respectively.

8.4.2 Underestimation Problem

As explained in the beginning of this chapter, in backfilling using system-generated predictions, the problem of underestimation can lead to an adverse impact on system

performance. Therefore, predictors need to carefully take care of this problem and reduce the number of underestimated jobs as much as possible. Our predictor notices this problem and considerably reduces the number of underestimated jobs compared with other predictors as shown in Table 8.4. This is since we add the standard deviation as in Eq. (8.8) when we calculate the estimate. Furthermore, it is also caused by the fact that we separate the roles of big jobs from small jobs as explained in Section 8.3.3 and take into account the percentage of underestimated jobs in the objective function when training for the best parameters (see Eq. (8.9)).

TABLE 8.4: Percentage of underestimated jobs.

Trace	Tsafrir's	Li's	Our Predictor
SDSC02	47%	50%	25%
SDSC04	45%	49%	29%
HPC2N04	55%	52%	36%
HPC2N05	51%	50%	31%

Reducing the number of underestimated jobs will certainly increase the number of jobs that are overestimated. However, in backfilling systems, overestimating jobs is still better than underestimating since this gives the scheduling algorithms some flexibility that leads to better schedules [73]. Furthermore, we also take care and try not to let the overestimation problem increase the prediction error by using the upper bound for an estimation as in Eq. (8.8). This mechanism really contributes efficiently to decrease the prediction error as shown in Sections 8.4.3 and 8.4.4.

8.4.3 The Mean Absolute Error

Results for the mean absolute error are shown in Table 8.5. Our predictor is clearly better than Tsafrir's in most cases. Comparing with Li's, our predictor is much better in case of SDSC04 but not much different in the other cases. In order to explain the reason, we draw Figure 8.2, which shows the cumulative distribution functions (CDF) of the runtimes. We would like to remind that our predictor will work well if there are several big jobs in a trace since we separate the roles of big jobs from small jobs as discussed in Section 8.3.3. This is demonstrated in case of SDSC04, where nearly 14% jobs are bigger than 10,000 seconds, our predictor is 20% better than Li's. Among all the traces, SDSC02 includes a large majority of small jobs (only 5% jobs are bigger than 10,000 seconds), therefore, our predictor yields approximately the same error as Li's in this case. Note, we already showed in Section 8.3.3 that only a small number of big jobs will also contribute to a considerable prediction error. Although HPC2N04 and HPC2N05 also have several big jobs, the results are not much different between all three predictors. It is because their mean runtimes are

very big (see Table 8.7). This means that these traces have a lot of very long jobs and all predictors including ours do not look far enough in the past to find similar jobs. Consequently, when predicting for very long jobs they find wrong similar jobs and yield big prediction errors. Particularly, our predictor is still a little bit better than Li's in case of HPC2N04 because its mean runtime is still not very big.

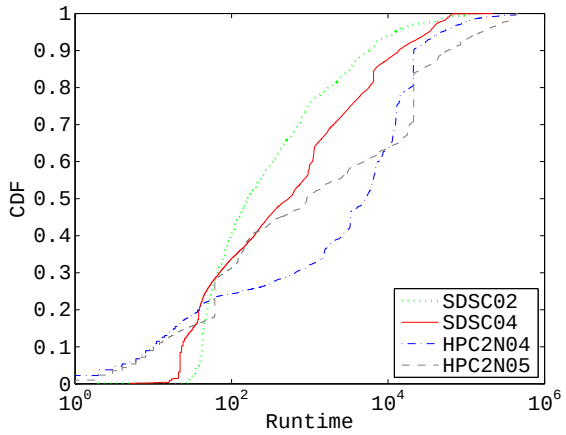


FIGURE 8.2: Runtime distribution of traces.

TABLE 8.5: Mean absolute error (in minutes).

Trace	Tsafrir's (1)	Li's (2)	Our Predictor	Compared with (1)	Compared with (2)
SDSC02	38.7	30.5	30.7	21%	0%
SDSC04	60.4	60.1	48.1	20%	20%
HPC2N04	187	164	156	17%	5%
HPC2N05	474	436	438	8%	0%

8.4.4 The Weighted Absolute Error

The study in [108] demonstrates that no single error metric can fully predict scheduling performance and the metric mean absolute error is not sufficient for characterizing errors. According to [108], the metric mean absolute error is even much worse than the metric weighted absolute error in capturing the adverse impact of runtime estimate errors on scheduling performance. With respect to the metric weighted absolute error, Table 8.6 shows that our predictor works considerably better than Tsafrir's and Li's.

It is easy to see that the best result is from HPC2N05 because this trace has a large number of big and long jobs (see Table 8.7). However, the difference is not strong in case of SDSC02 because most of its jobs are small and short (68% jobs are serial and mean runtime is only 52 minutes).

TABLE 8.6: Weighted absolute error (in minutes).

Trace	Tsafrir's (1)	Li's (2)	Our Predictor	Compared with (1)	Compared with (2)
SDSC02	523	402	374	29%	7%
SDSC04	328	339	271	17%	20%
HPC2N04	984	903	805	18%	11%
HPC2N05	2134	2543	1722	19%	32%

TABLE 8.7: Characteristics of traces.

Trace	Mean Runtime (minutes)	Serial Jobs	Parallel Jobs
SDSC02	52	68%	32%
SDSC04	86	33%	67%
HPC2N04	246	63%	37%
HPC2N05	478	35%	65%

8.5 Summary

In this chapter, we presented a novel approach to predict application runtimes in backfilling parallel systems. We also explained that both inaccuracy and underestimation in prediction are not good for backfilling systems. Hence, our predictor is suitable for backfilling scheduling because it obtained good accuracy and considerably reduced the number of underestimated jobs. The idea of our approach is to define job similarity by a combination of the categorization and the instance based learning methods, with a new similarity function (see Eq. (8.5) in Section 8.3.1). Moreover, another idea is to separate big jobs from small jobs (see Section 8.3.3). In future work, we will use this predictor to estimate job response times. Moreover, we also want to evaluate how well this approach really impacts the efficiency of backfilling scheduling.

Chapter 9

Conclusions

Scheduling plays a significant role in producing good performance for clusters and grids. Smart scheduling policies in these systems are essential to enable efficient resource allocation mechanisms. One of the key factors that have a strong effect on scheduling is the workload. This workload problem is associated with four research topics to obtain an effective scheduler, namely workload characterisation, workload modeling, performance evaluation and prediction, and scheduling design. Workload data collected from real systems are the best source for improving our knowledge about performance issues of clusters and grids. Observed features of these workloads are precious sources of clues, which can be utilized to enhance scheduling. To this end, several long-term parallel and grid workloads have been collected [31, 80] and this thesis used these real workloads in the study of workload characterisation, workload modeling, performance evaluation and prediction. Our research resulted in many workload modeling tools, a performance predictor and several useful clues that are essential to develop efficient cluster and grid schedulers.

First of all, this thesis provided a comprehensive characterisation on real cluster and grid workloads, with an emphasis on several statistical features. It was shown that the features long range dependence and temporal burstiness exist strongly in real system job traffic. The analysis also indicated the common presence of the characteristics temporal locality, cross-correlation and spatial burstiness of runtime and parallelism processes. In particular, an approach of quantifying spatial burstiness was suggested and demonstrated experimentally to work well. This efficient approach was used to quantify spatial burstiness in real world data and the results suggested that spatial burstiness deserves more attention from research community, so that correct workloads are used when evaluating scheduling algorithms. In addition, the well-known Bag-of-Tasks (BoT) behaviour was also analyzed and the result showed that users often submit a large number of their jobs under this behaviour. We identified several pattern structures of BoT arrivals and it turned out that the Generalized Pareto distribution is a good fit for BoT interarrivals. It was also illustrated that BoT arrivals are bursty and can have similar or completely contrary structures as job arrivals, with respect to long range dependence and periodicity. In addition to BoT arrivals, statistics like autocorrelation and cross-correlation were also applied to

BoT sizes, runtimes, parallelisms and estimates for a comprehensive analysis of this behaviour. With the common presence of all the mentioned workload features in real world data and their crucial roles on performance issues of clusters, grids and clouds, we argued that more research on workload modeling and performance evaluation is needed.

Once a particular feature is observed in real workloads, it will be interesting to discover its potential impact on scheduling performance. Hence, it is essential to develop representative workload models that can efficiently capture a particular feature. To this end, we have successfully developed a job arrival model with long range dependence and temporal burstiness. This model is based on the multifractal wavelet model (MWM) [46, 85]. MWM is a good choice when it comes to long-range autocorrelations but it should be adapted before applying to system job traffics because we showed that cluster and grid job arrivals do not only exhibit long range dependence but also show temporal burstiness. Experimental results showed that the developed job arrival model can accurately control the temporal burstiness degree and can produce long range dependence well. Moreover, the synthetic job arrival process generated by the model also fits the marginal distribution nicely.

With respect to the workload attributes runtime and parallelism, a new model was proposed to produce several important features such as temporal locality, spatial burstiness, cross-correlation, etc. Especially, these two workload attributes were not modeled separately but were modeled with the job arrival attribute to capture the well-known Bag-of-Tasks behaviour. Hence, this model can be considered as a comprehensive model since it can generate three workload attributes at the same time, which is necessary for simulation use. With a large number of jobs submitted as part of BoTs, it is clear that the temporal-spatial correlation in parallel system workloads is mainly due to this behaviour. Therefore by capturing BoTs, this model helps to study the impact of the temporal-spatial correlation on scheduling performance. Furthermore, this comprehensive representative workload model can be used in many other research aspects. It first helps to generate realistic workloads for the evaluation of newly designed scheduling algorithms. Secondly, the model can be used to evaluate the impact of individual workload characteristics on scheduling performance. Thirdly, because of the ability to capture many workload characteristics, the model is a useful tool for evaluation studies on the impact of interactions between workload characteristics on the performance of clusters and grids.

After the study of workload characterisation and modeling, performance evaluation was investigated in this thesis. Evaluating the performance impact of workloads on scheduling not only helps to enrich the understanding of a system but also provides several important clues that can improve schedulers. Although many workload features were shown to be present commonly in real world data and they certainly have potential impacts on scheduling, these impacts are hardly described in the literature. Therefore, performance issues of numerous workload characteristics, including long range dependence, temporal burstiness, temporal locality and the cross-correlation between runtimes and parallelisms, were investigated.

For long range dependence and temporal burstiness, our evaluation results for a single cluster showed that they have severe performance impacts and can significantly affect a scheduling design. In the scenario of a grid, we indicated that in particular situations, grid and cluster jobs may join together to improve scheduling performance. However in most cases, grid and background cluster jobs do not really affect each other. With regard to the correlation between runtime and parallelism, our experiments showed that a positive correlation has more serious impact on parallel system performance than a negative correlation. However, backfilling policies can help to reduce the impact. Furthermore, scheduling design results may differ significantly under different degrees of correlation. With respect to temporal locality, it decreases the scheduling performance of a parallel system but when the system is overloaded, the feature turns to be useful for non-backfilling policies. Therefore, we conclude that any efficient scheduling design should take long range dependence, temporal burstiness, temporal locality and the cross-correlation between runtime and parallelism into account for a good evaluation. To enable such designs, we introduced, besides the developed comprehensive representative workload model, an efficient procedure that can help to tune the degree of the cross-correlation on demand.

Although we found that a non-backfilling algorithm, that utilizes the temporal locality feature well, can be used when a system is overloaded, we also showed that if the system is underloaded, a backfilling policy that restrains the wait time of a job to a certain threshold is a better choice. Designing a particular backfilling policy is completely feasible because we developed an efficient runtime predictor with high accuracy in this thesis. This predictor takes into account the problems of under/over-estimations, which are important for backfilling because underestimating a job has it killed by the system while overestimating a job gives it less chance to be backfilled.

In addition to the research achievements summarised above, the results of our study have opened several research opportunities for future work, which are presented as follows:

1. Although the workload model introduced in this thesis is comprehensive for the simulation use of parallel scheduling because it provides adequately three workload attributes (including arrival time, runtime and number of processors), the model should include another attribute, namely the user estimated runtime, since this attribute is essential for backfilling scheduling. As the user estimated runtime attribute has a strong correlation with the other attributes, modeling it separately would be impractical and unrealistic. Instead, it should be modeled in conjunction with the Bag-of-Tasks behaviour and temporal locality because it has been demonstrated that similar jobs often exhibit similar estimates [105].

2. The importance of periodicity such as daily cycles is demonstrated in [22]. Therefore, this feature should be captured and integrated into the workload model.

3. Job failures/cancels are also interesting for consideration in any workload model [52]. The occurrence of a job failure/cancel event can release computing resources, stimulate schedulers to re-schedule jobs, and so apparently affect scheduling. Despite their importance, a model for job failures/cancels is still lacking in the literature.

4. Research on performance evaluation of workload features is still scarce in the literature. In particular, the behaviour of workload features on grids is an open area.

5. Though starting a long time ago, parallel system scheduling is a hard problem that is still actively researched now [26, 27]. Compared with cluster scheduling, grid scheduling is a broader subject [77]. Despite the large number of scheduling strategies for both clusters and grids, these strategies are often designed without taking into account performance impacts of workload features, and thereby miss important clues that can improve schedulers. Hence, scheduling design based on workload features is a promising research direction for future work.

Bibliography

[1] P. Abry, R. Baraniuk, P. Flandrin, R. Riedi, D. Veitch, "The Multiscale Nature of Network Traffic: Discovery, Analysis, and Modelling", IEEE Signal Processing Magazine, vol. 19, pp. 28-46, 2002.

[2] C. Anglano, M. Canonico, "Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications", Advances in Grid Computing, vol. 3470, pp. 630-639, 2005.

[3] H. Baghban, A. M. Rahmani, "A Heuristic on Job Scheduling in Grid Computing Environment", International Conference on Grid and Cooperative Computing, pp. 141-146, 2008.

[4] J. Beran, "Statistics for Long Memory Processes", Chapman and Hall, New York, 1994.

[5] R. Buyya, M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", Concurrency and Computation Practice and Experience, vol. 14, pp. 1175-1220, 2002.

[6] O. Cappe, E. Moulines, J. Pesquet, A. Petropulu, X. Yang, "Long-Range Dependence and Heavy-Tail Modeling for Teletraffic Data", IEEE Signal Processing Magazine, vol. 19, pp. 14-27, 2002.

[7] Catalina, http://www.sdsc.edu/catalina/.

[8] W. Cirne, F. Berman, "A Comprehensive Model of the Supercomputer Workload", IEEE Annual Workshop on Workload Characterization, pp. 140-148, 2001.

[9] S. H. Chiang, A. Arpaci-Dusseau, M. K. Vernon, "The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 2537, pp. 103-127, 2002.

[10] S. H. Chiang, M. K. Vernon, "Production Job Scheduling for Parallel Shared Memory Systems", International Parallel and Distributed Processing Symposium, pp. 10, 2001.

[11] M. D. de Assuncao, A. di Costanzo, R. Buyya, "Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters", International Symposium on High Performance Distributed Computing, pp. 141-150, 2009.

[12] P. J. Denning, "The Locality Principle", Communications of the ACM, vol. 48, pp. 19-24, 2005.

[13] A. B. Downey, "Predicting Queue Times on Space-Sharing Parallel Computers", International Parallel Processing Symposium, pp. 209-218, 1997.

[14] A. B. Downey, "Using Queue Time Predictions for Processor Allocation", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 1291, pp. 35-57, 1997.

[15] A. B. Downey, D. G. Feitelson, "The Elusive Goal of Workload Characterization", ACM SIGMETRICS Performance Evaluation Review, vol. 26, pp. 14-29, 1999.

[16] C. Dumitrescu, I. Raicu, I. Foster, "DI-GRUBER: A Distributed Approach to Grid Resource Brokering", ACM/IEEE Conference on Supercomputing, pp. 38, 2005.

[17] A. Erramilli, O. Narayan, W. Willinger, "Experimental Queueing Analysis with Long-Range Dependent Packet Traffic", IEEE/ACM Transactions on Networking, vol. 4, pp. 209-223, 1996.

[18] D. G. Feitelson, "Locality of Sampling and Diversity in Parallel System Workloads", International Conference on Supercomputing, pp. 53-63, 2007.

[19] D. G. Feitelson, "Workload Modeling for Computer Systems Performance Evaluation", Book Draft, Version 0.32, 2011.

[20] D. G. Feitelson, A. W. Mu'alem, "Utilization and Predictability in Scheduling the IBM SP2 with Backfilling", International Parallel Processing Symposium, pp. 542-546, 1998.

[21] D. G. Feitelson, B. Nitzberg, "Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 949, pp. 337-360, 1995.

[22] D. G. Feitelson, E. Shmueli, "A Case for Conservative Workload Modeling: Parallel Job Scheduling with Daily Cycles of Activity", IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 1-8, 2009.

[23] D. G. Feitelson, D. Tsafrir, "Workload Sanitation for Performance Evaluation", International Symposium on Performance Analysis of Systems and Software, pp. 221-230, 2006.

[24] W. Fischer, K. Meier-Hellstern, "The Markov-Modulated Poisson Process (MMPP) Cookbook", Performance Evaluation, vol. 18, pp. 149-171, 1993.

[25] E. Frachtenberg, D. G. Feitelson, "Pitfalls in Parallel Job Scheduling Evaluation", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 3834, pp. 257-282, 2005.

[26] E. Frachtenberg, U. Schwiegelshohn, "New Challenges of Parallel Job Scheduling", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 4942, pp. 1-23, 2008.

[27] E. Frachtenberg, U. Schwiegelshohn, D. Feitelson, L. Rudolph, Annual Workshops on Job Scheduling Strategies for Parallel Processing 1995-2011, http://www.cs.huji.ac.il/~feit/parsched/.

[28] C. Fraley, A. E. Raftery, "Model-Based Clustering, Discriminant Analysis, and Density Estimation", Journal of the American Statistical Association, vol. 97, pp. 611-631, 2002.

[29] R. Gibbons, "A Historical Application Profiler for Use by Parallel Schedulers", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 1291, pp. 58-77, 1997.

[30] D. E. Goldenberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989.

[31] Grid Workloads Archive, http://gwa.ewi.tudelft.nl/.

[32] L. He, S. A. Jarvis, D. P. Spooner, D. Bacigalupo, G. Tan, G. R. Nudd, "Mapping DAG-Based Applications to Multiclusters with Background Workload", International Symposium on Cluster Computing and the Grid, pp. 855-862, 2005.

[33] E. J. Heikkila, L. Hu, "Adjusting Spatial-Entropy Measures for Scale and Resolution Effects", Environment and Planning B: Planning and Design, vol. 33, pp. 845-861, 2006.

[34] High Performance Computing Center North, http://www.hpc2n.umu.se/.

[35] A. Horváth, M. Telek, "A Markovian Point Process Exhibiting Multifractal Behavior and Its Application to Traffic Modeling", International Conference on Matrix-Analytic Methods in Stochastic Models, pp. 183-208, 2002.

[36] H. E. Hurst, "Long Term Storage Capacity of Reservoirs", American Society of Civil Engineers, vol. 116, pp. 770-799, 1951.

[37] A. Iosup, C. Dumitrescu, D. Epema, H. Li, L. Wolters, "How are Real Grids Used? The Analysis of Four Grid Traces and Its Implications", International Conference on Grid Computing, pp. 262-269, 2006.

[38] A. Iosup, M. Jan, O. Sonmez, D. Epema, "The Characteristics and Performance of Groups of Jobs in Grids", International Euro-Par Conference on Parallel Processing, LNCS, vol. 4641, pp. 382-393, 2007.

[39] A. Iosup, O. Sonmez, S. Anoep, D. Epema, "The Performance of Bags-of-Tasks in Large-Scale Distributed Systems", International Symposium on High Performance Distributed Computing, pp. 97-108, 2008.

[40] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, J. Riodan, "Modeling of Workload in MPPs", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 1291, pp. 95-116, 1997.

[41] J. H. Jenkins, "Stationary Joint Distributions Arising in the Analysis of the M/G/1 Queue by the Method of the Imbedded Markov Chain", Journal of Applied Probability, vol. 3, pp. 512-520, 1966.

[42] M. A. Johnson, S. Narayana, "Descriptors of Arrival-Process Burstiness with Application to the Discrete Markovian Arrival Process", Journal of Queueing Systems, vol. 23, pp. 107-130, 1996.

[43] Y. K. Kwok, I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors", Journal ACM Computing Surveys, vol. 31, pp. 406-471, 1999.

[44] LCG, http://lcg.web.cern.ch/LCG/.

[45] H. Li, "Long Range Dependent Job Arrival Process and Its Implications in Grid Environments", International Conference on Networks for Grid Applications, 2007.

[46] H. Li, "Realistic Workload Modeling and Its Performance Impacts in Large Scale eScience Grids", IEEE Transactions on Parallel and Distributed Systems, vol. 21, pp. 480-493, 2010.

[47] H. Li, "Workload Characterization, Modeling and Prediction in Grid Computing", PhD Thesis, LIACS, Leiden University, 2007.

[48] H. Li, "Workload Dynamics on Clusters and Grids", Journal of Supercomputing, vol. 47, pp. 1-20, 2009.

[49] H. Li, R. Buyya, "Model-Based Simulation and Performance Evaluation of Grid Scheduling Strategies", Future Generation Computer Systems, vol. 25, pp. 460-465, 2009.

[50] H. Li, D. Groep, L. Wolters, "Mining Performance Data for Meta-scheduling Decision Support in the Grid", Future Generation Computer Systems, vol. 23, pp. 92-99, 2007.

[51] H. Li, D. Groep, L. Wolters, "Workload Characteristics of a Multi-Cluster Supercomputer", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 3277, pp. 176-193, 2005.

[52] H. Li, D. Groep, L. Wolters, J. Templon, "Job Failure Analysis and Its Implications in a Large-Scale Production Grid", International Conference on e-Science and Grid Computing, pp. 27, 2006.

[53] H. Li, M. Muskulus, "Analysis and Modeling of Job Arrivals in a Production Grid", ACM SIGMETRICS Performance Evaluation Review, vol. 34, pp. 59-70, 2007.

[54] H. Li, M. Muskulus, L. Wolters, "Modeling Correlated Workloads by Combining Model Based Clustering and a Localized Sampling Algorithm", International Conference on Supercomputing, pp. 64-72, 2007.

[55] H. Li, L. Wolters, "An Investigation of Grid Performance Predictions through Statistical Learning", Workshop on Tackling Computer System Problems with Machine Learning Techniques, 2006.

[56] H. Li, L. Wolters, "Towards A Better Understanding of Workload Dynamics on Data-Intensive Clusters and Grids", International Parallel and Distributed Processing Symposium, pp. 1-10, 2007.

[57] V. Lo, J. Mache, K. Windisch, "A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 1459, pp. 25-46, 1998.

[58] LSF, http://www.platform.com/.

[59] U. Lublin, D. G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs", Journal of Parallel and Distributed Computing, vol. 63, pp. 1105-1122, 2003.

[60] J. S. Maritz, "Distribution-Free Statistical Methods", Chapman and Hall/CRC, Second Edition, 1995.

[61] Maui, http://www.clusterresources.com/products.php.

[62] MCLUST, http://www.stat.washington.edu/mclust/.

[63] E. Medernach, "Workload Analysis of a Cluster in a Grid Environment", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 3834, pp. 36-61, 2005.

[64] T. N. Minh, Homepage, http://www.liacs.nl/home/minhtn/.

[65] T. N. Minh, L. Wolters, "Model-Driven Simulation to Evaluate Performance Impact of Workload Features on Parallel Systems", International Conference on Cluster Computing, pp. 84-92, 2011.

[66] T. N. Minh, L. Wolters, "Modeling Job Arrival Process with Long Range Dependence and Burstiness Characteristics", International Symposium on Cluster Computing and the Grid, pp. 324-330, 2009.

[67] T. N. Minh, L. Wolters, "Modeling Parallel System Workloads with Temporal Locality", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 5798, pp. 101-115, 2009.

[68] T. N. Minh, L. Wolters, "Performance Impact of Job Arrivals on Clusters and Grids through Realistic Model-Based Simulation", International Symposium on Performance Evaluation of Computer and Telecommunication Systems, pp. 22-29, 2011.

[69] T. N. Minh, L. Wolters, "Towards a Profound Analysis of Bags-of-Tasks in Parallel Systems and Their Performance Impact", International Symposium on High Performance Distributed Computing, pp. 111-122, 2011.

[70] T. N. Minh, L. Wolters, "Using Historical Data to Predict Application Runtimes on Backfilling Parallel Systems", Euromicro International Conference on Parallel, Distributed and Network-Based Processing, pp. 246-252, 2010.

[71] T. N. Minh, L. Wolters, D. Epema, "A Realistic Integrated Model of Parallel System Workloads", International Conference on Cluster, Cloud and Grid Computing, pp. 464-473, 2010.

[72] H. H. Mohamed, D. Epema, "KOALA: A Co-Allocating Grid Scheduler", Concurrency and Computation: Practice and Experience, vol. 20, pp. 1851-1876, 2008.

[73] A. W. Mu'alem, D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling", IEEE Transactions on Parallel and Distributed Systems, vol. 12, pp. 529-543, 2001.

[74] R. R. Muntz, J. E. G. Coffman, "Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems", Journal of the ACM, vol. 17, pp. 324Ũ338, 1970.

[75] J. L. Myers, A. D. Well, R. F. Lorch Jr, "Research Design and Statistical Analysis", Third Edition, 2010.

[76] J. Myung, "Tutorial on maximum likelihood estimation", Journal of Mathematical Psychology, vol. 47, pp. 90-100, 2003.

[77] J. Nabrzyski, J. M. Schopf, J. Weglarz, "Grid Resource Management: State of the Art and Future Trends", Kluwer Academic Publisher, 2003.

[78] National Center for Supercomputing Applications, http://www.ncsa.uiuc.edu/.

[79] NIST/SEMATECH, "E-Handbook of Statistical Methods", http://www.itl.nist.gov/div898/handbook/, 2006.

[80] Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/.

[81] Performance Data Miner, http://www.liacs.nl/home/hli/pdm/.

[82] K. Ranganathan, I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications", International Symposium on High Performance Distributed Computing, pp. 352-358, 2002.

[83] V. J. Ribeiro, R. H. Riedi, M. S. Crouse, R. G. Baraniuk, "Simulation of nonGaussian Long-Range-Dependent Traffic using Wavelets", International Conference on Measurement and Modeling of Computer Systems, pp. 1-12, 1999.

[84] R. H. Riedi, "Long Range Dependence: Theory and Applications", Editors Doukhan, Oppenheim, Taqqu, 2002.

[85] R. H. Riedi, M. S. Crouse, V. J. Ribeiro, R. G. Baraniuk, "A Multifractal Wavelet Model with Application to Network Traffic", IEEE Transactions on Information Theory, vol. 45, pp. 992-1018, 1999.

[86] S. M. Ross, "Introduction to Probability Models", Academic Press, 8th Edition, 2003.

[87] B. K. Ryu, A. Elwalid, "The Importance of Long-Range Dependence of VBR Video Traffic in ATM Traffic Engineering: Myths and Realities", ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 3-14, 1996.

[88] San Diego Supercomputer Center, http://www.sdsc.edu/.

[89] L. J. Senger, M. J. Santana, R. H. C. Santana, "An Instance-Based Learning Approach for Predicting Execution Times of Parallel Applications", International Information and Telecommunication Technologies Symposium, pp. 9-15, 2005.

[90] C. E. Shannon, W. Weaver, "The Mathematical Theory of Communication", University of Illinois Press, 1963.

[91] Slurm, https://computing.llnl.gov/linux/slurm/documentation.html.

[92] W. Smith, I. Foster, V. Taylor, "Predicting Application Run Times Using Historical Information", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 1459, pp. 122-142, 1998.

[93] W. Smith, V. Taylor, I. Foster, "Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 1659, pp. 202-219, 1999.

[94] W. Smith, P. Wong, "Resource Selection Using Execution and Queue Wait Time Predictions", NAS Technical Report Number: NAS-02-003, NASA Ames Research Center, 2002.

[95] B. Song, C. Ernemann, R. Yahyapour, "Parallel Computer Workload Modeling with Markov Chains", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 3277, pp. 47-62, 2004.

[96] S. Song, K. Hwang, Y. K. Kwok, "Trusted Grid Computing with Security Binding and Trust Integration", Journal of Grid Computing, vol. 3, pp. 53-73, 2005.

[97] M. S. Squillante, D. D. Yao, L. Zhang, "The Impact of Job Arrival Patterns on Parallel Scheduling", ACM SIGMETRICS Performance Evaluation Review, vol. 26, pp. 52-59, 1999.

[98] S. Srinivasan, R. Kettimuthu, V. Subramani, P. Sadayappan, "Selective Reservation Strategies for Backfill Job Scheduling", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 2537, pp. 55-71, 2002.

[99] P. Tan, M. Steinbach, V. Kumar, "Introduction to Data Mining", Addison Wesley Publishing, 2005.

[100] M. S. Taqqu, W. Willinger, R. Sherman, "Proof of a Fundamental Result in Self-Similar Traffic Modeling", Computer Communication Review, vol. 27, pp. 5-23, 1997.

[101] M. S. Taqqu, V. Teverovsky, W. Willinger, "Estimators for Long-Range Dependence: An Empirical Study", Fractals., vol. 3, pp. 785-798, 1995.

[102] The Distributed ASCI Supercomputer 3, http://www.cs.vu.nl/das3/.

[103] TOP500 Supercomputing Sites, http://www.top500.org/.

[104] D. Tsafrir, Y. Etsion, D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates", IEEE Transactions on Parallel and Distributed Systems, vol. 18, pp. 789-803, 2007.

[105] D. Tsafrir, Y. Etsion, D. G. Feitelson, "Modeling User Runtime Estimates", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 3834, pp. 1-35, 2005.

[106] D. Tsafrir, D. G. Feitelson, "Instability in Parallel Job Scheduling Simulation: the Role of Workload Flurries", International Conference on Parallel and Distributed Processing, pp. 54, 2006.

[107] R. G. Turcott, M. C. Teich, "Interevent-Interval and Counting Statistics of the Human Heartbeat Recorded from Normal Subjects and Patients with Heart Failure", Ann. Biomed. Eng. 24, pp. 269-293, 1996.

[108] S. Vasupongayya, S. H. Chiang, "Performance Problems of Using System-Predicted Runtimes for Parallel Job Scheduling", International Conference on Parallel and Distributed Computing and Systems, pp. 369-374, 2007.

[109] S. Verboven, P. Hellinckx, J. Broeckhove, F. Arickx, "Dynamic Grid Scheduling Using Job Runtime Requirements and Variable Resource Availability", International Euro-Par Conference on Parallel Processing, LNCS, vol. 5168, pp. 223-232, 2008.

[110] M. Wang, A. Ailamaki, C. Faloutsos, "Capturing the Spatio-Temporal Behavior of Real Traffic Data", Performance Evaluation, vol. 49, pp. 147-163, 2002.

[111] J. Weinberg, "Job Scheduling on Parallel Systems", Job Scheduling Strategies for Parallel Processing", 2002.

[112] C. Weng, X. Lu, "Heuristic Scheduling for Bag-of-Tasks Applications in Combination with QoS in the Computational Grid", Future Generation Computer Systems, vol. 21, pp. 271-280, 2005.

[113] Y. Zhang, H. Franke, J. E. Moreira, A. Sivasubramaniam, "An Integrated Approach to Parallel Scheduling Using Gang-Scheduling, Backfilling, and Migration", Job Scheduling Strategies for Parallel Processing, LNCS, vol. 2221, pp. 133-158, 2001.

[114] Y. Zhang, H. Franke, J. E. Moreira, A. Sivasubramaniam, "Improving Parallel Job Scheduling by Combining Gang Scheduling and Backfilling Techniques", International Parallel and Distributed Processing Symposium, pp. 133-142, 2000.

[115] J. Zhang, B. S. Lee, X. Tang, C. K. Yeo, "Improving Job Scheduling Performance with Parallel Access to Replicas in Data Grid Environment", Journal of Supercomputing, vol. 56, pp. 245-269, 2009.

[116] J. Zilber, O. Amit, D. Talby, "What is Worth Learning from Parallel Workloads?: A User and Session Based Analysis", International Conference on Supercomputing, pp. 377-386, 2005.

Samenvatting

Slimme scheduling policies voor clusters en grids zijn essentieel voor efficiënte mechanismen voor resource-allocatie. De workload is een van de belangrijkste factoren die een sterk effect op scheduling hebben. Er zijn vier onderzoeksonderwerpen die verband houden met het workload-probleem: karakterisatie van workload, modellering van workload, evaluatie en predictie van performance, en ontwerp van scheduling. Verschillende lange-termijn workloads voor parallele systemen en grids zijn bij elkaar gebracht, en dit proefschrift heeft gebruik gemaakt van deze lange-termijn workloads bij de studie van de vier genoemde onderwerpen. Ons onderzoek heeft geresulteerd in een aantal modelleer-tools, een performance-voorspeller, en verscheidene nuttige aanwijzingen die essentieel zijn bij het efficient ontwikkelen van schedulers voor clusters en grids.

Dit proefschrift heeft allereerst een uitgebreide karakterisatie verschaft voor workloads van echte clusters en grids, met nadruk op verschillende statistische kenmerken. Er is aangetoond dat de kenmerken *long range dependence* en *temporal burstiness* zeer sterk voorkomen in het echte verkeer van systeem-jobs. De analyse heeft ook aangegeven dat de karakteristieken *temporal locality*, *cross-correlation* en *spatial burstiness* gezamenlijk voorkomen. Daar komt nog bij dat het bekende *Bag-of-Tasks*-gedrag ook is geanalyseerd, waarbij het resultaat liet zien dat gebruikers vaak een groot aantal van hun jobs volgens dit gedrag aanbieden. Met het gemeenschappelijk voorkomen van alle genoemde workload-kenmerken in echte data, en hun cruciale rol bij zaken die performance van clusters, grids en clouds betreffen, hebben we beargumenteerd dat meer onderzoek naar het modelleren van workload en evaluatie van performance nodig is.

Zodra een specifiek kenmerk wordt waargenomen in een echte workload, wordt het interessant om de mogelijke impact op scheduling te ontdekken. Het is daarom essentieel om representatieve workload-modellen te ontwikkelen, die efficiënt een specifiek kenmerk kunnen vangen. Daartoe hebben we succesvol een aankomst-model voor jobs ontwikkeld met *long range dependence* en *temporal burstiness*. Experimentele resultaten hebben laten zien dat het ontwikkelde model nauwkeurig de mate van *temporal burstiness* kan beheersen, en dat het *long range dependence* goed kan produceren. Bovendien past het door het model gegenereerde kunstmatige aankomst-proces ook bij de marginale verdeling.

Met betrekking tot de workload-eigenschappen runtijd en parallellisme is een voorstel voor een nieuw model gemaakt dat verschillende belangrijke kenmerken kan

produceren, zoals *temporal locality*, *spatial burstiness*, *cross-correlation*, etc. Meer in het bijzonder zijn deze twee workload-eigenschappen niet apart gemodelleerd, maar samen met de job-aankomst-eigenschap ten einde het *Bag-of-Tasks*-gedrag te vangen. Dit model kan dientengevolge beschouwd worden als een uitgebreid model, aangezien het drie workload-eigenschappen tegelijk kan genereren, wat nodig is voor gebruik bij simulatie. Dit workload-model kan gebruikt worden bij verschillende onderzoeksaspecten. Het helpt ten eerste om realistische workloads te genereren voor de evaluatie van nieuwe scheduling-algoritmen. Ten tweede kan het model gebruikt worden om de impact van individuele workload-karakteristieken op de performance van scheduling te evalueren. En ten derde, vanwege de mogelijkheid om veel workload-karakteristieken te vangen, is het model een nuttig tool voor evaluatie-studies van de wisselwerking tussen workload-karakteristieken op de performance van clusters en grids.

Na de studie van karakterisatie en modellering van workloads, is in dit proefschrift de evaluatie van performance onderzocht. Aspecten van de performance van talloze workload-karakteristieken zijn onderzocht, waaronder *long range dependence*, *temporal burstiness*, *temporal locality* en *cross-correlation*. Voor *long range dependence* en *temporal burstiness* laten onze evaluatie-resultaten voor een enkele cluster zien dat ze een belangrijke invloed hebben op de performance en dat ze een significante invloed hebben op een scheduling-ontwerp. Voor het scenario van een grid hebben we aangegeven dat in bijzondere situaties, grid-jobs en cluster-jobs samengevoegd kunnen worden om de performance van de scheduling te verbeteren. In de meeste gevallen hebben daarentegen grid-jobs en achtergrondscluster-jobs geen echte invloed op elkaar. Met betrekking tot *cross-correlation* hebben onze experimenten laten zien dat een positieve correlatie een meer serieuze invloed op de performance van het parallelle systeem heeft dan een negatieve correlatie. *Backfilling* policies kunnen echter helpen de impact te verkleinen. Verder kunnen ontwerpresultaten voor de scheduling significant verschillen bij verschillende mate van correlatie. Voor *temporal locality* geldt dat de performance van de scheduling bij een parallel systeem verlaagt, maar wanneer het systeem overvol is, wordt het kenmerk nuttig voor *non-backfilling* policies. We concluderen daarom dat elk efficiënt ontwerp voor scheduling rekening moet houden met *long range dependence*, *temporal burstiness*, *temporal locality* en *cross-correlation* om goede evaluatie te bereiken. Om zulke ontwerpen mogelijk te maken, hebben we naast het ontwikkelde uitgebreide representatieve workload-model, een efficiënte procedure geintroduceerd die kan helpen om de mate van *cross-correlation* naar believen in te stellen.

Hoewel we gevonden hebben dat een *non-backfilling* algoritme, dat goed het kenmerk *temporal locality* benut, gebruikt kan worden wanneer het systeem overvol is, hebben we ook laten zien dat als het systeem onderbeladen is, een *backfilling* policy die de wachttijd van een job beperkt tot een zekere drempelwaarde een betere keuze is. Het ontwerpen van een zekere *backfilling* policy is zeer goed mogelijk omdat we in dit proefschrift een efficiënte runtijd-voorspeller met hoge precisie hebben ontwikkeld. Deze voorspeller houdt rekening met de problemen van onderschatting en overschatting, die belangrijk zijn voor *backfilling*.

Acknowledgements

I have experienced four years in Leiden, as a PhD student, with lots of memories, where I received many kind helps from colleagues, friends, and my family members. Therefore, I would like to thank them since I probably could not have finished my research well without their helps.

I am grateful to Thieu, Mattias, Kristian, Gerard, Walter, Vian, Saskia, for their kind supports on commenting my thesis, interesting discussion and social life.

I would like to thank my Vietnamese friends, who gave me unforgettable parties that made my life in the Netherlands more exciting and relaxing.

I express my special thanks to my parents and my younger sister. I always feel comfortable and motivated every time I see their dear face. Last but not least, my most special thanks are dedicated to Thuy, my darling wife, for sharing my difficulties and giving me an invaluable present: Gia Khuong, our lovely son.

Curriculum Vitae

Tran Ngoc Minh was born on December 15, 1982, in Dong Nai, Vietnam. He received the BEng degree in computer engineering in 2005 from Ho Chi Minh University of Technology (HCMUT), Vietnam. Since 2005 he was recruited as a young lecturer at HCMUT. At the same time, he pursued and obtained the MSc degree in computer science in 2007, also from HCMUT. In the summer of 2007, he was offered a DAAD scholarship for short-term research at the Interdisciplinary Center for Scientific Computing, University of Heidelberg, Germany. Since April 2008, he joined the Leiden Institute of Advance Computer Science at Leiden University, the Netherlands, where he pursued and obtained a PhD degree in computer science on March 2012. His research interests are in the areas of resource management and scheduling, particularly in performance analysis, modeling and evaluation of large-scale parallel and distributed systems.