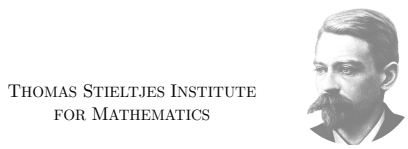
Robust Applications in Time-Shared Distributed Systems

Menno Dobber

Robust Applications in Time-Shared Distributed Systems

Robust Applications in Time-Shared Distributed Systems



The research described in this thesis has been supported by the Netherlands Organisation for Scientific Research (NWO), and the Thomas Stieltjes Institute for Mathematics.

ISBN: 90-8659036-5
The aquarelles on the cover are painted by Corinne Langerhuizen.
The cover is composed by Friso Dobber and HenkJan Dobber.

Printed by PrintPartners Ipskamp B.V., the Netherlands.

VRIJE UNIVERSITEIT

Robust Applications in Time-Shared Distributed Systems

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op maandag 13 november 2006 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Anton Menno Dobber

geboren te Naarden

promotoren: prof.dr. G.M. Koole
prof.dr. R.D. van der Mei

PREFACE

The last four years have been very interesting in many respects and a great experience which I will never forget. The PhD project has given me the opportunity to travel to India, Singapore, France, and Wales, which was very enjoying and a nice possibility to spread the research results. Over the years, I have met many interesting people from everywhere and learned a lot about different cultures and social behavior. Moreover, the teaching of students was a fascinating experience. I am thankful to a number of people.

First of all, my special thanks go to my promotors Ger Koole and Rob van der Mei who gave me this opportunity and supervised me in a excellent way.

Next, I would like to thank the reading committee, which consists of Henri Bal, Dick Epema, Geurt Jongbloed, Zhen Liu, and Philippe Nain, for reading my thesis and providing me detailed feedback.

In addition, I appreciated the people of the research groups 'Optimization of Business Processes' and 'High Performance Distributed Computing', who created a pleasant environment to work in and assisted me if necessary. In particular, I am grateful to Mathijs den Burger, Henri Bal, Thilo Kielmann, Ger Koole, and Rob van der Mei, because of the pleasing cooperation in the NWO research project. Furthermore, I mention Auke Pot and Maarten Soomer who have become my best friends. We had a lot of fun and spent much time together in and outside the university. Moreover, I give thanks to Tom van der Schaaf for the interesting discussions about our research projects and other subjects.

Subsequently, I want to thank a few relatives. First, I thank my parents HenkJan and Saskia, and my brothers Onno and Friso for their interest in my work and their support. HenkJan and Friso were of great help in designing and composing the cover in cooperation with artist Corinne Langerhuizen. In addition, I acknowledge my grandpa Jan, who is very enthusiastic about my work, and great-aunt Elly, who showed much interest in the mathematical part of my research.

Finally, I would like to address the important role of Amée during the past few years. Because of her unconditional love and support she provided the optimal basis to accomplish my PhD thesis.

Menno Dobber,
Amsterdam, 2006

Committee:

prof.dr.ir. H.E. Bal	(Vrije Universiteit, Amsterdam)
dr.ir. D.H.J. Epema	(Technische Universiteit, Delft)
dr.ir. G. Jongbloed	(Vrije Universiteit, Amsterdam)
dr. Z. Liu	(IBM T. J. Watson Research Center, Hawthorne, USA)
dr. P. Nain	(INRIA, Sophia Antipolis, France)

Faculty of Sciences
Department of Mathematics
De Boelelaan 1081a
1081HV Amsterdam
The Netherlands

TABLE OF CONTENTS

1

INTRODUCTION

The emergence of computer and information technology has changed our modern society dramatically. The use of computers has become widespread and their processing power has increased tremendously. At the same time, the advances in high-speed networking have enabled computers to collaborate. This has created a tremendous source of processor power, which has opened up many possibilities for running advanced computation-intensive applications within a reasonable time frame. Typical examples of such applications are in computational fluid dynamics (aerospace, military), electro-magnetic simulations, scheduling problems that use neural networks or permutations, environmental modeling (earth, ocean, atmospheric simulations), environmental phenomenology, economic modeling, image processing, health-care fraud, and market segmentation. See [37] for an overview of computation-intensive applications.

However, the sources of computer power are typically shared by different users or applications. Consequently, the available amount of processing power is often highly dynamic, hard to predict and often unreliable. This raises the need for methods to effectively cope with those uncertain factors. In this thesis, we focus on *the development of techniques that make the applications robust against the ever-changing processor speeds in global-scale time-shared computing environments.*

Robust applications are commonly defined as applications that (1) cope with connecting and disconnecting resources to the environment, (2) are insensitive to fluctuations in the available processor speeds, and (3) deal with the network dynamics. In this thesis, we focus on the development of methods that deal with the fluctuations in the resource speeds and use the general word 'robust' for this type of robustness.

1.1 Clusters and Grid environments

Since the emergence of computers and communicating technologies, networks of distributed computers have been created to execute computation-intensive applications. The development of combined computers has been expanded [35, 36] and many processor-intensive applications have been deployed on collaborating processors. Processor networks range from local-scale clusters of computers (e.g., the DAS environment [1]) to global-scale distributed grids, which are connected by the Internet, such as

the Planetlab [2]. These two environments are fundamentally different. A main advantage of running parallel programs on clusters is their predictability: the resources are homogeneous, the processing speeds are static and known beforehand, and the availability of resources is based on reservation which leads to a guaranteed amount of processing capacities. A disadvantage is the high cost of those clusters. As an alternative, one can use the idle cycles available on processor-shared globally distributed grids. Main advantages are the relatively low cost and the potentially unlimited resources. A grid environment, however, is highly unpredictable in many respects: resources have different and usually unknown capacities, they can be added and removed at any time, and the processing speeds fluctuate over time due to the often implemented processor sharing concept, where multiple jobs execute simultaneously on a single processor. As a consequence, applications that perform well in a cluster environment may perform badly when executed in a grid environment. These observations raise the challenge to develop techniques that make parallel programs suitable for execution in a large-scale grid environment. In this context, it is challenging to achieve good performance of parallel applications running in a grid environment.

1.2 Grid testbeds

Over the years, much research has been done on grid computing. Initially, in the absence of publicly available grid environments, the research on how to cope with fluctuations on grid nodes was mainly theoretically oriented. Recently, however, a variety of grid testbeds have been developed (e.g., EuroGRID [3], Gridlab [4], Planetlab [2]). This enables us to perform extensive experiments with grid applications and comprehensive measurements, to investigate how well grid applications perform in practice, and how they can be improved.

1.3 Single Program Multiple Data

To be able to run a program or computation on multiple processors at the same time it is necessary to make the program suitable for running in parallel. Programs that have this property are called parallel programs. Parallel programs can be applied for different purposes, such as parallel computing and virtual environments. The computations and communications structure of many parallel applications that focus on computing can be described by the Single-Program-Multiple-Data (SPMD) model [33]. Within SPMD programs each processor runs the same program, but uses its own data. SPMD programs have the property that the problem can be divided into sub-problems or jobs (i.e., the problem space can be divided into parts) each of which can be solved or executed in roughly the same way.

In a SPMD program, each run consists of a number of iterations (I) that consist of the execution of P jobs that are distributed on P processors: each processor receives one job per iteration. Every run contains I synchronization moments: after computing the jobs, all the processors send their data and wait for each others data before the next iteration starts. In general, the run time equals the maximum of the individual iteration times. Figure 1.1 presents the situation for one iteration of a SPMD run in a grid environment. The figure shows that each processor receives a job and the iteration

time equals the maximum of the individual job runtimes plus the synchronization time. Equal load balancing (ELB) assumes no prior knowledge of processor speeds of the nodes, and consequently balances the load equally among the different nodes. The standard SPMD program is implemented according to the ELB principle.

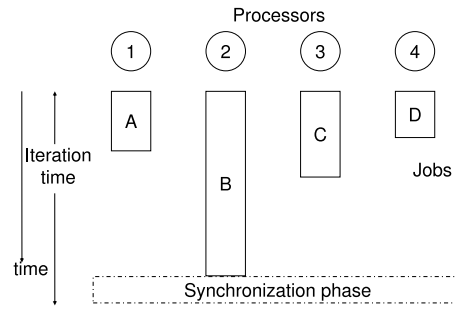


Figure 1.1: Illustration of a standard Single-Program-Multiple-Data Application on four nodes

Currently, not many of the SPMD-type of applications are able to run in a grid environment due to the fact that they cannot deal with the ever-changing environment. Especially, the synchronization in SPMD programs causes inefficiency: one late job can delay the whole process. This raises the need for methods that make the SPMD applications robust against changes in the grid environment.

1.4 Fluctuations in Grid environments

The increasing popularity of parallel applications in a grid environment creates many new challenges regarding the performance of grid applications, e.g., in terms of running times. To this end, it is essential to reach a better understanding of (1) the nature of fluctuations in processing speeds and the relevant time scale of these fluctuations, (2) the impact of the fluctuations on the running times of grid applications, (3) the forecasting methods that accurately predict those processing speeds, and (4) effective means to cope with the fluctuations: the load-distribution strategies. Below, we provide an overview of the state-of-the-art in the research that has been done on the different aspects. In the remainder of this section, we discuss the literature about (1)-(3), and in Section 1.5 we get into detail about the types of load-distribution strategies.

1.4.1 Statistical properties

As described above, key characteristics of a global-scale grid are the strong burstiness in the amount of load on the resources and on the network capacities, and the fact that processors may be appended to or removed from the grid at any time. To cope with these characteristics, it is essential to develop techniques that make applications robust against the dynamics of the grid environment. For these techniques to be effective, it is important to have an understanding of the statistical properties of the dynamics of a grid environment. Today, however, the statistical properties of the dynamic behavior of real global-scale grid environments are not well understood.

In the literature, a significant number of papers have been devoted to data analysis of different properties of grids or networks. Three types of grid-property investigations can be distinguished: (1) a complete focus on the investigation of one grid property: for example, the statistical characteristics of network arrivals [60], of availability [56], and of load [20]; (2) an exploration on the statistical properties of a grid property and followed by simulation studies to address different types of questions, varying from grid design questions (e.g., is a global grid feasible?) to basic questions (e.g., what scheduling strategies are needed?). Examples are the investigations on the statistical properties of life times of UNIX processes to develop load balancing strategies [41, 51], and investigations on a characterization of the availability of desktop grids to explore the affection on its utility [46], and (3) research on the statistical characteristics of a grid property to develop a prediction method: those research steps have been done for load to predict total run times of applications [62], for load to predict the load [61], and for the throughput to predict the throughput [15, 54]. Consequently, the final step is to use these predictions to develop a dynamic load-balancing or scheduling algorithm (see [22] with its preceding papers [20, 21, 61], and [17] in combination with [72]). Despite the fact that many papers focus on the statistical characteristics of grid properties, no papers concentrate on the running times of consecutive tasks on shared processors of a grid.

Research has been performed on the relation between processor load and the running times by Dinda et al. [21, 23]. Although these two factors in theory are closely related, they find that in practice it is hard to relate the running times and the load, because many other factors (e.g., memory space) also have an influence on the running times. For that reason, load and running times of tasks may have strongly different characteristics, and it is necessary to investigate the running time characteristics.

1.4.2 Impact on running times

Fluctuations in the available resources (e.g., computing power, bandwidth) are known to have an impact on the running times of parallel applications. Over the past few decades, performance of parallel applications has received much attention in the research community. Due to the difficulty of analyzing realistic variations, most of the fluctuations were imitated and therefore controllable (see for example [10]), while performance experiments were performed in a controllable cluster. However, the variations in grid environments are not manageable, which limits the applicability of these results in a real grid environment. In the research community several groups focus on performance aspects of grid applications. Alternatively, mathematicians typically build stochastic models to describe the performance of resources, the network and dependencies related to runs of parallel programs. They create algorithms to decrease running times, and analyze these algorithms mathematically [6, 7, 38, 43, 44, 53, 63, 69, 76]. Such a mathematical approach may be effective in some cases; however, usually unrealistic assumptions have to be made to provide a mathematical analysis, which limits the applicability of the results. On the other hand, computational grid experts develop well-performing strategies for computational grids. However, due to the difference in fluctuations between general grid environments and computational grids, the effectiveness of these strategies in a grid environment is questionable [57]. A third group of

researchers focuses on large-scale applications with parallel loops (i.e., loops with no dependencies among their iterations) [9, 13, 42, 65], combining the development of strategies based on a probabilistic analysis with experiments on computational grids with regulated load. However, due to the absence of dependencies among the iterations of those applications, these strategies are not applicable to parallel applications with those dependencies. These observations stress the importance for an *integrated* analysis of grid applications, combining the three above-mentioned approaches, to analyze properties, dependencies and distributions within a grid environment, and to implement the ideas in a real grid environment and verify how the methods perform in practice.

1.4.3 Prediction methods

In the literature, a significant number of papers have been devoted to prediction methods. The prediction schemes can be broadly classified into two main categories: linear and non-linear predictors. For grid environments, several linear models have been proven successful for predicting future properties of a grid: Exponential Smoothing (ES) for predicting running times of jobs ([64]), Autoregressive models (AR) for predicting the load [23] and network traffic ([61]), Linear Regression (LR) for predicting total running times of parallel applications ([50]), and a tendency-based predictor ([74]), also for predicting the load. Furthermore, the Network Weather Service (NWS) prediction algorithm that selects between different linear predictors (e.g. [71, 72, 73]) predicts different grid entities, such as CPU availability. Moreover, in other research areas interesting predictors have been developed with possible applications in grid environments. As described in [66], several adaptive ES-based methods (e.g., Trigg and Leach [67], Whybark [70], Mentzer [55], and Pantazopoulos and Pappis [58]) have been developed, which have shown to be very accurate in predicting, for example, economic and societal quantities. Currently, the applicability in grid environments of non-linear prediction models, like Neural Networks, has not been investigated.

To predict the running times on the basis of the measured load, Dinda et al. [21, 23] analyze the relation between processor load and the running times of jobs. They conclude that predicting the running times by the load is impracticable, due to the influence of other aspects (e.g., hardware mechanisms) on those times, and those are hard, or even technically impossible, to gather in practice. The prediction of running times is even further complicated by the fact that in a real grid environment even the load (i.e., CPU utilization) often can not be measured at all. To circumvent this problem, we focus on methods to predict future running times *only* on the basis of the *past running times* of jobs, not requiring any additional - and possibly unavailable - measurement data.

In grid environments, if prediction methods for job runtimes are used to trigger load re-balancing actions they should be simple and fast. Moreover, since the monitoring capabilities in a real grid environment will be at best limited, prediction methods should be based on only a small number of measurement parameters.

1.5 Types of load-distribution strategies

Contrary to the implementations that are based on Equal Load Balancing (ELB) three methods for parallel applications have been developed to deal with the fluctuations in processor speeds on the nodes: Static Load Balancing (SLB), Dynamic Load Balancing (DLB), and Job Replication (JR). In this section, we discuss the details about the four different load-distribution strategies.

1.5.1 Equal Load Balancing

This is a logical default implementation, because it assumes no prior knowledge about the processor speeds or other grid properties. In this type of implementations the load is distributed equally among the different nodes in the resource set. Above in Figure 1.1 an example is shown of an ELB iteration.

1.5.2 Static Load Balancing

Static Load Balancing (SLB) strategies first use a number of ”cold iterations” to estimate the average processor speeds or to perform other relevant measurements. Next, during a load rescheduling phase this information is used to compute the load distribution such that the total expected running time of the application is minimized. Subsequently, the load is redistributed according to this distribution. Finally, the implementation performs the rest of the iterations and the load distribution remains constant.

1.5.3 Dynamic Load Balancing

The third method to deal with changing processor speeds is to implement Dynamic Load Balancing (DLB) schemes. DLB adapts the load on the different processors in proportion to the expected processor speeds. This technique strongly relies on the effectiveness of prediction methods: significant speedups can be obtained by good scheduling schemes based on accurate predictions, as is demonstrated in an experimental setting in [12, 17, 22, 52].

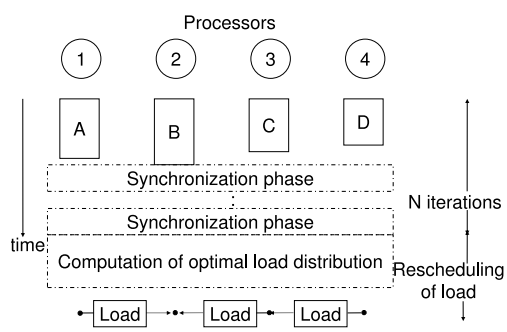


Figure 1.2: Illustration of Dynamic Load Balancing on four nodes

DLB starts with the execution of an iteration, which does not differ from the common SPMD program explained above. However, at the end of each iteration the processors predict their processing speed for the next iteration. We select one processor to

be the DLB scheduler. After every N iterations the processors send their prediction to this scheduler. Subsequently, this processor calculates the "optimal" load distribution given those predictions and sends relevant information to each processor. The load distribution is optimal when all processors finish their calculation exactly at the same time. Therefore, it is "optimal" when the load assigned to each processor is proportional to its predicted processor speed. Finally, all processors redistribute the load. Figure 1.2 provides an overview of the different steps within a DLB implementation on 4 processors. The effectiveness of DLB partly relies on the possibilities of partitioning the load.

Load balancing at every single iteration is rarely a good strategy. On the one hand, the running time of a parallel application depends directly on the overhead of DLB, and therefore it is better to increase the number of iterations between two successive load balancing steps. On the other hand, less load balancing leads to an imbalance of the load for the processors for sustained periods of time, due to significant changes in processing speeds. For these reasons, it is necessary in an DLB implementation to find an appropriate number of iterations between two balancing steps.

1.5.4 Job Replication

The last method for parallel applications that have been developed to deal with the fluctuations in processor speeds on the nodes is job replication (JR) [5, 8, 16, 18, 39, 47, 59, 48]. Moreover, [14] focuses on the queue waiting times of jobs that run on processors without processor sharing and concludes that for his setting redundant job requests decreases the times. Generally, JR makes a given number of copies of each job, sends the copies and the original job to different processors, and waits until the first replication is finished.

In a R-JR run, $R-1$ exact copies of each job have been created and have to be executed, such that there exist R samples of each job. Two copies of a job perform exactly the same computations: the datasets, the parameters, and the calculations are completely the same. If one of these properties of two jobs is different, we call the jobs unequal. A JR run consists of I iterations. One iteration takes in total R steps. R copies of all P jobs have been distributed to P processors and therefore, each processor receives during each iteration R different jobs. As soon as a processor has finished one of the copies it sends a message to the other processors that they can kill the job and start the next job in the sequence. The number of synchronization moments I is the same as for the non-JR case. At least one sample of each of the P different jobs has to be finished before the iteration is finished.

Figure 1.3 shows the situation for a 2-JR run on four processors. Each job and its copy are distributed to $R = 2$ processors and during one iteration each processor receives two jobs. Processor one finished as first job A and sends a 'finalize' message to processor two. Sending the message over the Internet takes some time and, therefore, it takes a while before the other processors start the next job. Each job-type time, which is the duration of a specific job type (the original and its copies), equals the minimum of all its job runtimes plus a possible send time. An individual processor time of one iteration equals the sum of the job-type times which were sent to that processor and the send times of the 'kill'-messages. Finally, the iteration time of all the processors

corresponds to the sum of the synchronization time and the maximum of all processor times.

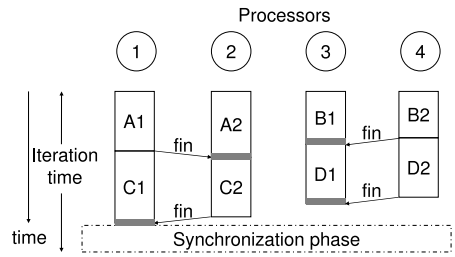


Figure 1.3: Illustration of two times Job Replication on four nodes

1.6 Types of analyses

To investigate the effectiveness of the different types of load-distribution strategies, analyses and comparisons have to be performed. Generally, three types of investigations can be applied to accurately verify whether certain algorithms or strategies are effective: (1) mathematical calculation, (2) trace-driven simulation, and (3) real implementation. Mathematical calculations have the advantage compared to the other two types that (1) these provide a tremendous insight in the factors that have the most impact on certain quantities, and (2) formulas can be derived that represent the situation for many different values of the parameters. Disadvantages are that (1) they take a lot of time to model the situation and derive formulas for the key quantities, and (2) often many assumptions have to be made to develop a model that can be used to perform computations, which make the model less realistic. Therefore, it is necessary to make a well-founded decision if mathematical calculations have to be performed to get insight in the situation or one of the other two types of analyses.

Next, we compare the trace-driven simulation with the real implementation as analyzing method. On the one hand, trace-driven simulations have the advantage that (1) they are often easier to implement than real implementations because, for example, no advanced communication implementations are necessary, and (2) less clock time is needed to test different experimental setups. Consequently, trace-driven simulations need a shorter time period in which more situations can be analyzed. For example, the investigations in Chapter 4 show the performance gain of a real implementation of DLB in a real grid environment. As many as 60 days of parallel-implementation runs were necessary in order to derive the performance improvement (speedup) of DLB compared to Equal Load Balancing (ELB) on four processors. The trace-driven simulations in this chapter take less time and more analyses can be performed. On the other hand, analysis of the durations of the processes in an application have to be made to develop realistic simulations. To this end, in this thesis we program a real implementation of DLB to acquire more knowledge about the durations of the different processes within an DLB application. In turn this knowledge is used in trace-driven simulations.

1.7 Overview of the thesis

In Chapter 2, the grid testbed and the grid application are described that are used to collect measurements. In addition, definitions that will be used throughout the whole thesis are introduced. Moreover, the details of the simulations of Static Load Balancing (SLB), Dynamic Load Balancing (DLB), and Job Replication (JR) are provided.

Second, in Chapter 3 the statistical properties are investigated of job runtimes on processors on the basis of the datasets that are collected in the preceding chapter. To this end, Box-and-Wisker plots, histograms, Auto Correlation Functions, Hurst parameters, and 17 other statistics have been analyzed. Furthermore, the relations between the 17 different statistics is investigated.

Next, in Chapter 4, an experimental analysis is provided of the performance of grid applications based on ELB, SLB and DLB in a global-scale grid environment on the basis of comprehensive trace-driven simulations and real implementation experiments. Moreover, the impact on the DLB run-times of the number of measurements between two load rescheduling steps and of the communication to computation are investigated.

Subsequently, in Chapter 5 a model is defined to compute the expected iteration times and speedups of SPMD programs that apply job replication (JR) on a set of homogeneous processors. In order to make estimations with the model a number of approximations that can easily be applied in the model computations are presented. Subsequently, six different sets of model assumptions ranging from less realistic and mathematically simple to realistic and mathematically difficult are applied in the computations and the results are analyzed. Moreover, trace-driven simulations of JR on homogeneous and on heterogeneous nodes are performed, based on real grid testbed measurements. Furthermore, the impact of different factors is analyzed.

In Chapter 6, the effectiveness of ELB, DLB and JR are compared by the results of trace-driven simulations. A comparison of the performance of those two methods on a heterogeneous globally distributed grid environment has never been performed. Subsequently, in-depth analysis shows the identification of an easy-to-measure statistic Y and a corresponding threshold value Y^* such that DLB consistently outperforms JR for $Y > Y^*$, whereas JR consistently performs better for $Y < Y^*$. This observation naturally leads to a simple and easy-to-implement approach that can make on-the-fly decisions about whether to use to DLB or JR. Elaborate simulations show that this new approach always performs at least as good as both DLB and JR in all circumstances. As such, the new approach provides a highly effective means to make parallel applications robust in large-scale grid environments.

Chapter 7 focuses on the development of a new prediction method for job runtimes on shared processors. To this end, the weak and strong points of several existing methods are analyzed on the basis of both theoretical and statistical analysis of experimental testbed data from a preceding chapter. In addition, a new prediction method, called Dynamic Exponential Smoothing (DES) is developed. The accuracy of the predictions resulting from DES are compared to that of the other prediction methods. Furthermore, the relation between the quality of the DES predictor and the statistical properties of the datasets is investigated.

Finally, in Chapter 8 a number of topics for further research are addressed.

The results presented in the remainder of this thesis are based on the following papers:

1. A. M. Dobber, G. M. Koole, and R. D. van der Mei. Dynamic load balancing for a grid application. In *Proceedings of the 11th International Conference on High Performance Computing (HiPC '04)*, pages 342–352. Bangalore, India, December 19-22, 2004. Springer-Verslag.
2. A. M. Dobber, G. M. Koole, and R. D. van der Mei. Dynamic load balancing experiments in a grid. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, pages 123–130. Cardiff, Wales, UK, May 9-12, 2005. IEEE Computer Society.
3. A. M. Dobber, R. D. van der Mei, and G. M. Koole. Statistical properties of task running times in a global-scale grid environment. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '06)*, pages 150–153. Singapore, May 16-19, 2006. IEEE Computer Society.
4. A. M. Dobber, R. D. van der Mei, and G. M. Koole. Effective prediction of job processing times in a large-scale grid environment. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC '06) (poster)*, pages 359–360. Paris, France, June 19-23, 2006. IEEE Computer Society.
5. A. M. Dobber, R. D. van der Mei, and G. M. Koole. Prediction method for job runtimes on shared processors: survey, statistical analysis and new avenues. In revision for *Performance Evaluation*.
6. A. M. Dobber, R. D. van der Mei, and G. M. Koole. Dynamic load balancing and replicating in a global-scale grid environment: a comparison. Submitted.
7. A. M. Dobber, G. M. Koole, R. Righter, and R. D. van der Mei. Scheduling jobs on homogeneous processors. In preparation.

2

EXPERIMENTAL SETUP

2.1 Introduction

As outlined in the previous chapter, in this thesis we perform analyses and experiments on different types of implementations in order to make parallel applications robust in grid environments. To this end, many datasets are needed. In this chapter, we explain the data collection details of all the datasets that are gathered. We describe in Section 2.2 which grid testbed and which nodes are used to collect data. In addition, in Section 2.3 we present the used grid application and in Section 2.4 which measurements are performed. In Section 2.5, we introduce definitions that are used throughout the whole thesis. Finally, we provide the details of the simulations of Static Load Balancing (SLB) in Section 2.6, of Dynamic Load Balancing (DLB) in Section 2.7, and of Job Replication (JR) in Section 2.8.

2.2 Global-scale distributed testbed: Planetlab

To carry out experiments with parallel applications in a realistic setting, the testbed must have the following key characteristics of a grid environment: (1) processor capacities often fluctuate over time, (2) the nodes work according to the processor shared principle, (3) processor loads change over time, (4) processors are geographically distributed, and (5) network conditions are hard to predict. A commonly used grid testbed environment that meets all these requirements is Planetlab [2]. Planetlab is an open, processor-shared globally distributed network for developing and testing planetary-scale network services. At the time of our experiments, version 2.0 of the Planetlab software was installed on the nodes. Planetlab is not a grid in a strict sense due to the fact that Globus [34], which consists of an integrated set of basic grid services, does not run on this network. Nevertheless, the expectations are that Planetlab will use the concepts of Globus and consequently that it will act according to the concepts of a processor-shared grid. For this reason, we nevertheless use the word grid throughout the thesis, even if we discuss the results of Planetlab experiments.

In order to investigate and gather reliable results of different implementations, settings and cases, it is necessary to perform experiments on many different Planetlab nodes. In total 22 nodes, which are single processors at different locations, were used

many times for the experiments in this thesis. Table 2.1 provides an overview of all the 22 globally distributed nodes that were used. In the next section we describe the application that has been used to collect data on these nodes.

Institution	City	Country	Abbreviation
Vrije Universiteit	Amsterdam	The Netherlands	ams
Tsinghua University	Beijing	China	china
Boston University	Boston	USA	boston
University of Cambridge	Cambridge	UK	cam
Datalogisk Institut	Copenhagen	Denmark	dk
Inria	Le Chesnay	France	inria
Technical University of Madrid	Madrid	Spain	mad
Moscow State University	Moscow	Russia	mos
California Institute of Technology	Pasadena	USA	caltech
University of Utah	Salt Lake City	USA	utah
University of California	San Diego	USA	sandiego
University of California	Santa Barbara	USA	santab
Seoul National University	Seoul	South Korea	seoul
Equinix	Singapore	Singapore	sing
University of Technology	Sydney	Australia	au
Ben-Gurion University of the Negev	Tel Aviv	Israel	telaviv
Academica Sinica	Taipei	Taiwan	tw
National Taiwan University	Taipei	Taiwan	ntu
University of Arizona	Tucson	USA	ar
University of British Columbia	Vancouver	Canada	ca
Warsaw University of Technology	Warsaw	Poland	warsch
University of Washington	Washington DC	USA	wash

Table 2.1: Overview of nodes used in the experiments

2.3 Successive Over-Relaxation

The application has also been carefully chosen so as to meet several requirements. The application must have the same dependencies between its iterations as a SPMD program, the structure of the dependencies should be simple, and it must have the possibility to adapt the load on the processors. A suitable application is the Successive Over Relaxation (SOR) application. SOR is an iterative method that has proven to be useful in solving Laplace equations [31]. Our implementation of SOR deals with a two-dimensional discrete state space $M \times N$, a grid. Each point in the grid has four neighbors, or less when the point is on the border of the grid. Mathematically this amounts to taking a weighted average of the values of the neighbors and its own value. The parallel implementation of SOR is based on the Red/Black SOR algorithm [40]. The grid is treated as a checkerboard and each iteration is split into phases, red and black. During the red phase only the red points of the grid are updated. Red points only have black neighbors, and no black points are changed during the red phase. During the black phase, the black points are updated in a similar way.

Using the Red/Black SOR algorithm, the grid can be partitioned among the avail-

able processors in several ways. We partitioned the grid stripe wise: each processor receives a set of columns. All processors can update different points of the same color in parallel. This update takes time, referred to as the calculation time. Before a processor starts the update of a certain color, it exchanges the border points of the opposite color with its neighbors. This amount of time is referred to as the send time. The total duration of an iteration is called the iteration time. Figure 2.1 illustrates the use of SOR over different processors.

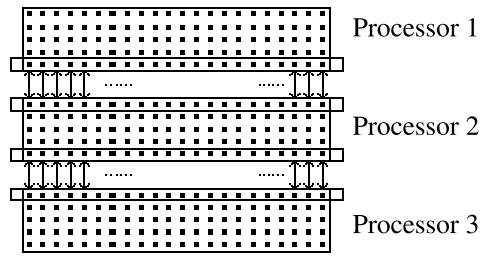


Figure 2.1: Parallel distribution in Successive Over Relaxation

2.4 Data collection

Throughout the whole thesis, many trace-driven simulations are performed in order to investigate the performance of several types of implementations and diverse settings. Trace-driven simulations are simulations that use traces of measurements instead of distributions. For those simulations to be realistic, it is necessary to investigate the properties of the key processes in SPMD programs, the dependencies between them, and to collect data about the durations of those processes.

To this end, we implement a real SPMD program, which is described below in this chapter, and, moreover, the different implementation types Equal Load Balancing (ELB), SLB, DLB, and JR, which are described in Chapter 1. Subsequently, we divide the whole run into sub-processes (e.g., job runtimes, send times) and measure their durations in order to gain insight into the most significant sub-processes during a SPMD program run. As a result, we are able to measure those sub-processes and develop realistic simulations. We conclude from the measurements that (1) all implementations at least contain the following types of phases that have a substantial impact on the total runtime: computation and synchronization phase, (2) in JR, also the duration of sending a finalize message between processors takes a considerable amount of time, and (3) in SLB and DLB, the same holds for the rescheduling phase. The other sub-processes have negligible durations. The computation phase consists of individual job runs. Within the synchronization phase all the processors send the new data to the other processors. Therefore, in order to simulate the computation and the synchronization phase it is reasonable to use data about the job and the send times. Summarizing, in order to perform realistic simulations, we need to collect data about the job runtimes, send times in the rescheduling phase, rescheduling times, and measurements of the durations of sending finalize messages. Below, we describe how we collected data about those times.

Job Runtimes

In order to gather enough data about the job runtimes to perform all the different experiments, in this thesis a total of I runs have been performed on the J different heterogeneous processors (see Table 2.1) on Planetlab [2]. Each run consists of the execution of K consecutive and identical jobs (or computations), and generates a dataset of K job runtimes (i.e., wall-clock times). Hence, we suppose the processing times (i.e., the actual processor times that the jobs take) of all the jobs to be constant and measure the time the jobs take on shared processors (i.e., the job runtimes). The average duration of a run is about two hours. We observed significant differences in the durations, some even exceed 10 hours. We did not run other applications on the same nodes during our runs to create changing load on the processors. We constructed the jobs such that on a completely available 500 megahertz processor, the job computations would take 2500 ms, which is a realistic job size in parallel applications. The time between successive runs that are performed on the same processor ranges from one day to one month. We notice that the more time between the runs, the more difference between the characteristics of the job runtimes of those runs. In order to correlate the datasets in the simulations, each run is started at 9:00 CET. Unfortunately, Planetlab version 2.0 was not mature enough at the time of the experiments to be able to run experiments on 130 different processors. However, the job runtimes of runs that are performed on the same node mostly show different characteristics (for more details we refer to Chapter 3). For these reasons, in many simulations we use those different datasets as if they were performed on two different homogeneous nodes at the same site. In order to work with the different job-runtime measurements, we define $JTM_i(k)$ as the kth job-runtime measurement of dataset i, where $i = 1, \ldots, I$ and $k = 1, \ldots, K$.

Send Times

For our simulations, we need realistic measurements of the synchronization times (STs) for the simulations of ELB, DLB, and JR. We discovered that the synchronization time strongly depends on the maximum of the send times between all pairs of neighbor processors. Therefore, for realistic simulations we need send times measurements between all possible pairs of nodes. Analysis of the send times have shown that the send times between two nodes do not depend on the number of other processors in the application that send data at the same time. Consequently, in total $J(J-1)/2(P-1)$ original SOR-application runs on $P = 4$ processors were necessary to generate datasets of the send times between each possible pair of nodes. In total $J(J-1)/2$ datasets of K send times have been created during this process. The send times are on average around 750 ms. We define $SndT_{i,j}(k)$ as the kth send time measurement between node i and j, where $i = 1, \ldots, J$, $j = 1, \ldots, i-1$, and $k = 1, \ldots, K$.

Sending Finalize-Messages Times

Further, it is important for the JR simulation to gather realistic measurements of the time that the finalize message takes to be sent from the fastest node to another node. Therefore, we implemented this send process in the SOR-application. This process is different from the above send process, because less information has to be sent, and no acknowledgement is needed. We ran again in total $J(J-1)/2(P-1)$ original SOR-application runs on four processors to generate $J(J-1)/2$ datasets of K finalize-message times. The finalize message times were on average around 300 ms. We define $FM_{i,j}(k)$ as the kth measurement of the finalize-message send-time between node i and j, where $i = 1, \ldots, J$, $j = 1, \ldots, i-1$, and $k = 1, \ldots, K$.

Rescheduling Phase Times

Moreover, for the DLB simulation it is essential to gather measurements of the rescheduling times. The steps to be taken in the DLB rescheduling phase: (1) the nodes send their prediction to the scheduler, (2) the scheduler computes the optimal load distribution, and (3) the nodes redistribute their load. We implemented the complete rescheduling phase in the DLB application. Analyses have shown that it is sufficient to randomly select in the simulation rescheduling times ($RSchT$s) from a set of L measurements. The $RSchT$ depends on too many different factors to subdivide the $RSchT$s to all those factors. In our setting, the total rescheduling procedure takes on average around 37500 ms. We note that it is possible to apply more effective packaging methods in this procedure, which can significantly decrease the $RSchT$s. Define $RSchT(l)$ as the lth measurement of the rescheduling time, where $l = 1, \ldots, L$.

In addition, we use the data of the DLB rescheduling times to estimate the overhead of the on-the-fly switches between the implementation types DLB and JR. Those times will in practice show comparable characteristics for the following reason. During an implementation switch, the steps to be taken are the same as during as DLB rescheduling phase. A switch from JR to DLB takes less time than a DLB rescheduling phase because the amount of the to be redistributed load is smaller; the nodes already contain most of the necessary load of the other processors. Small experiments have shown that a switch from JR to DLB takes around 60% of the time of a DLB rescheduling step. During a switch from DLB to JR, more data has to be redistributed due to the fact that all the processors need to gather replications of load from the other processors. This switch takes around 140% of the time of a DLB rescheduling step.

Taking everything together, we generated the following datasets for our trace-driven simulation analyses: $JTM_i(k)$, with $i = 1, \ldots, I$, $k = 1, \ldots, K$, $SndT_{i,j}(k)$, $FM_{i,j}(k)$, where $i = 1, \ldots, J$, $j = 1, \ldots, i-1$, $k = 1, \ldots, K$, and $RSchT(l)$ with $l = 1, \ldots, L$. In the settings of this thesis, $I = 130$, $J = 22$, $K = 2000$, and $L = 10000$.

2.5 Definitions

In this section, we define the variables, parameters, functions that will be used consequently through the thesis. We define N in SLB as the number of iterations at the beginning of a run that has been used to estimate the average processor speeds, in order to balance the load, and in DLB as the number of iterations between two load rescheduling phases. Furthermore, we define f in DLB as the prediction method that has been used to predict the job runtimes. Moreover, we define t as the time in seconds that the computation of one iteration would take when it has been executed on one completely available 500 megahertz Pentium 2 processor. This t indicates the problem size. The default t value, $t_{default}$ in the runs performed to gather the data of this thesis 2.5 s. Finally, we define P as the number of processors in the resource set. All those P processors receive a proportion of the total load. Given the above parameters, we define:

$E(t,P)$:= expected run time of a SPMD program run of size t with ELB on P processors,

$E_{it}(t,P)$:= expected iteration time of a SPMD program of size t with ELB on P processors,

$S(N,t,P)$:= expected run time of a SPMD program run of size t with SLB on P processors, which balances load after N iterations,

$S_{it}(N,t,P)$:= expected iteration time of a SPMD program of size t with SLB on P processors, which balances load after N iterations,

$D(N,f,t,P)$:= expected running time of a SPMD program run of size t with DLB and predictor f on P processors, which balances load every N iterations,

$D_{it}(N,f,t,P)$:= expected iteration time of a SPMD program run of size t with DLB and predictor f on P processors, which balances load every N iterations,

$R(t,R,P)$:= expected running time of a SPMD program run of size t with R-JR on P processors,

$R_{it}(t,R,P)$:= expected iteration time of a SPMD program of size t with R-JR on P processors.

As stated above, $E(t,P)$ is the expected running time of an ELB run and $S(0,t,P)$ indicates the expected running time of SLB run that uses no 0 iterations to balance the load at the beginning of the run. Furthermore, $D(\infty,f,t,P)$ equals the expected running time of a DLB run that balances every ∞ iterations, and $R(t,1,P)$ is the expected running time of a 1-JR run (i.e., each job exists only one time). Those running times mean the same and, therefore, we notice that: $E(t,P) = S(0,t,P) = D(\infty,f,t,P) = R(t,1,P)$.

In addition, we define for each implementation type (i.e., ELB, SLB, DLB, and JR) the theoretically lowest possible running times. The exact computations of these times are described further in this chapter. We define the following lowest possible running times:

$$\begin{aligned} S^*(t,P) &:= \text{expected running time of a SLB run of size } t \text{ on } P \text{ processors that} \\ &\quad \text{optimally balances the load at the beginning of the run,} \\ D^*(t,P) &:= \text{expected running time of a DLB run of size } t \text{ that optimally} \\ &\quad \text{balances the load every iteration,} \\ R^*(t,P) &:= \text{expected running time of the JR run of size } t \text{ on } P \text{ processors} \\ &\quad \text{with the optimal } R. \end{aligned}$$

To compare the performance under different strategies, we define the speedups of those different implementations as the number of times those strategies are faster than a ELB run with the same parameters t and P:

$$\begin{aligned} \text{speedup } S(N,t,P) &:= \frac{E(t,P)}{S(i,t,P)} = \frac{E_{it}(t,P)}{S_{it}(i,t,P)} , \\ \text{speedup } D(N,f,t,P) &:= \frac{E(t,P)}{D(i,f,t,P)} = \frac{E_{it}(t,P)}{D_{it}(i,f,t,P)} , \\ \text{speedup } R(t,R,P) &:= \frac{E(t,P)}{R(t,R,P)} = \frac{E_{it}(t,P)}{R_{it}(t,R,P)} , \\ \text{speedup } S^*(t,P) &:= \frac{E(t,P)}{S^*(t,P)} , \\ \text{speedup } D^*(t,P) &:= \frac{E(t,P)}{D^*(t,P)} , \\ \text{speedup } R^*(t,P) &:= \frac{E(t,P)}{R^*(t,P)} . \end{aligned}$$

In all the simulations that are performed in this thesis, we make the following assumptions.

Assumption 2.5.1: The relation between the job size and the job runtimes is linear and satisfies the additivity and the homogeneity property.

Assumption 2.5.2: The relation between the amount of data which is sent between processors and duration of this send procedure is linear and satisfies the additivity and the homogeneity property.

These are commonly applied assumptions and mean that if two job sizes differ with a factor a, the job runtimes also differ with a factor a or that if the amount of data which is sent increases with a factor a, the send time also increases with a factor a.

Furthermore, we define the following variables for $k = 1, \ldots, K$.

$$\begin{aligned} JT_i(k) &:= k\text{th rescaled job-runtime measurement on processor } i \\ &= \frac{t}{t_{default}} \times \frac{JTM_i(k)}{P}, \\ \hat{y}_i(k) &:= \text{prediction of } JT_i(k), \\ EJT_i(k) &:= \text{effectively simulated job-runtime on processor } i \text{ in iteration } k, \\ IT(k) &:= k\text{th iteration time}, \\ ST(k) &:= \text{synchronization time in iteration } k \ = \max_{i,j=1,\ldots,P} SndT_{ij}(k). \end{aligned}$$

The $t/t_{default}$ in the formula for $JT_i(k)$ indicates the proportion of the job size compared to the job size of the measurements of the original runs. The formula contains the factor P^{-1} because each processor in a run with P processors receives this proportion of the total load.

The communication-to-computation ratio (CCR) indicates the durations of the total send times compared to the total computation times. We define the CCR as the total send times in an average ELB run with two processors divided by the total job runtimes. Consequently, the definition that is used in this thesis is as follows.

Definition 2.5.3:

$$CCR := \frac{1}{J(J-1)/2} \sum_{i=1}^{J} \sum_{j=1}^{i-1} \frac{\sum_{k=1}^{K} SndT_{i,j}(k)}{\sum_{k=1}^{K} JT_i(k)}.$$

The CCR in our datasets is approximately 0.01. Since the CCR is linear related to the job sizes we derive the following formula of the CCR for a given job size t.

Definition 2.5.4:

$$CCR(t) := \frac{t_{default}}{t} 0.01, \text{ for } t, t_{default} > 0.$$

This equals in our setting $0.025/t$, because $t_{default} = 2.5$ which is described in the beginning of 2.5.

Let X be a stochastic variable and $x_1, \ldots, x_n$ a sequence of n samples from X. Then, we define the following unbiased estimations of the mean, median, and the standard deviation.

Definition 2.5.5: The expectation of the stochastic variable X, $\mathbb{E}(X)$, can be estimated by the unbiased sample mean, $\overline{X}$:

$$\mathbb{E}(X) \approx \overline{X} := \frac{1}{n} \sum_{i=1}^{n} x_i,$$

Definition 2.5.6: The median of the stochastic variable X can be estimated by

$$median((x_1, \ldots, x_n)) \quad := \quad \begin{cases} y_{(n+1)/2} & \text{if } n \text{ is odd,} \\ \frac{1}{2}(y_{n/2} + y_{n/2+1}) & \text{if } n \text{ is even,} \end{cases}$$

with

$$y_1 := \min((x_1, \ldots, x_n)), \ldots, y_n := \max((x_1, \ldots, x_n)).$$

Definition 2.5.7: The standard deviation of stochastic variable X, $\sigma(X)$, can be estimated by the sample standard deviation, which is an unbiased estimation:

$$\sigma(X) \approx \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mathbb{E}(X))^2}.$$

In addition, the following statistics can be defined.

Definition 2.5.8: The variance of stochastic variable X, $Var(X)$, has the following definition:

$$Var(X) = \sigma^2(X) := (\sigma(X))^2 = \mathbb{E}(X - \mathbb{E}(X))^2.$$

Definition 2.5.9: The coefficient of variation, c_X, of a non-negative stochastic X with expectation above 0 is defined as:

$$c_X := \frac{\sigma(X)}{\mathbb{E}(X)}.$$

Next, we define the correlation coefficient, $r_{X,Y}$, between two stochastic variables X, and Y. The correlation coefficient as defined here has the official name Pearson product-moment correlation coefficient. This is a statistical measure that quantifies the linear dependency between two variables. The lower bound of the correlation coefficient is -1, which indicates the strongest possible negative linear dependency. A correlation of 0 indicates no linear dependency. The upper bound equals 1, which shows a strong positive linear dependency between the variables.

Definition 2.5.10: The correlation coefficient, $r_{X,Y}$, between two variables X, and Y, is defined as

$$r_{X,Y} := \frac{\mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)}{\sigma(X)\sigma(Y)}.$$

Given the above definitions, we are able to define the following formula for the variance of the sum of two variables.

Property 2.5.11: Given two variables X, and Y, the variance of the sum of these variables, $\sigma^2(X+Y)$ is defined as

$$\sigma^2(X+Y) = \sigma^2(X) + \sigma^2(Y) + 2r_{X,Y}\sigma(X)\sigma(Y).$$

Furthermore, we define the following measures that indicate the error or deviation between two sequences of data values $x_1, \ldots, x_n$, and $y_1, \ldots, y_n$.

Definition 2.5.12: The mean absolute difference/error, MAD or MAE, is defined as:

$$MAE := \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i|.$$

Definition 2.5.13: The mean squared difference/error of a sequence, MSD or MSE, can be computed by the following formula:

$$MSE := \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2.$$

Definition 2.5.14: The square root of the mean squared error, $RMSE$, equals the square root of the MSE.

$$RMSE := \sqrt{MSE}.$$

2.6 Simulations of Static Load Balancing

In this section we describe the simulation details of SPMD programs that use SLB. First, we describe the simulations of $S(i, t, P)$, and second how we compute the $S^*(t, P)$. We use the definitions of Section 2.5.

2.6.1 Running times

The following steps have been incorporated in the simulations:

Step 1: Randomly select a resource set $S = \{p_1, \ldots, p_P\}$ of P processors from the datasets of the job runtimes, generated in Section 2.4, and order them such that the send times, measured in Section 2.4, between the processors are minimized. These numbers $p_1, \ldots, p_P$ correspond to the numbers of the datasets.

Step 2: Compute the average job runtimes of each processor of the first N iterations, which is an estimation of the expected job runtimes:

$$\frac{1}{N} \sum_{m=1}^{N} JT_i(m). \tag{2.1}$$

As a result, the expected speed of processor i, estimated by the fraction of the total load processed per ms, for the iterations after the first N is approximately

$$\frac{1}{P\frac{1}{N}\sum_{m=1}^{N} JT_i(m)}. \tag{2.2}$$

Further, the expectation of the total processor speed of the P processors together is approximately the sum of those speeds:

$$\sum_{i=1}^{P} \frac{1}{P\frac{1}{N}\sum_{m=1}^{N} JT_i(m)}. \tag{2.3}$$

Hence, the expected time of the next iteration without send and rescheduling times given an optimal load distribution is approximately

$$\frac{1}{\sum_{i=1}^{P} \frac{1}{P\frac{1}{N}\sum_{m=1}^{N} JT_i(m)}}. \tag{2.4}$$

Define for SLB runs the optimal load fraction to be the proportion of the total load that a processor receives such that its job-runtime average equals the average of the job runtimes of the other processors given that they all receive the optimal fraction of the load. As a consequence, the optimal load fraction of processor i is approximately

$$\frac{1}{\frac{1}{N}\sum_{m=1}^{N} JT_i(m) \sum_{i=1}^{P} \frac{1}{P\frac{1}{N}\sum_{m=1}^{N} JT_i(m)}} = \frac{P}{\sum_{m=1}^{N} JT_i(m) \sum_{i=1}^{P} \frac{1}{\sum_{m=1}^{N} JT_i(m)}}. \tag{2.5}$$

Finally, to simulate the $EJT_i(k)$, for $k = 1, \ldots, K$ with K as the number of iterations, of processor i for the next iterations, multiply the fractions with the real corresponding data values $JT_i(k)$ from the datasets.

$$EJT_i(k) = \begin{cases} \frac{JT_i(k)P}{\sum_{m=1}^{N} JT_i(m) \sum_{i=1}^{P} \frac{1}{\sum_{m=1}^{N} JT_i(m)}} & \text{if } k > N, \\ JT_i(k) & \text{if } k \leq N. \end{cases} \tag{2.6}$$

Step 3: Next, derive the $IT(k)$ for $k = 1, \ldots, K$ which is the maximum of the $EJT_i(k)$s of that iteration plus the $ST(k)$:

$$IT(k) = \max_{i=1,\ldots,P} EJT_i(k) + ST(k). \tag{2.7}$$

Step 4: Derive the running time of the SLB run by repeating step 3 K times, sum up all the iteration times, and add up a sample of the load rescheduling times:

$$S(N, t, P) := \sum_{k=1}^{K} IT(k) + RSchT(1), \tag{2.8}$$

with N as the number of iterations between two load rescheduling steps.

Step 5: Derive the expected running time of a SLB run on P processors by repeating steps 1 to 4 1000 times, and finally compute the average of the running times. Experimentation showed that 1000 as the number of repetitions is high enough in order to derive stable and reproducible results.

2.6.2 Optimal Static Load Balancing

We performed the following steps to compute the $S^*(t, P)$:

Step 1: Randomly select a resource set $S = \{p_1, \ldots, p_P\}$ of P processors from the datasets of the job runtimes, generated in Section 2.4, and order them such that the send times, measured in Section 2.4, between the processors are minimized. These numbers $p_1, \ldots, p_P$ correspond to the numbers of the datasets.

Step 2: The average job-runtime of processor i for a complete run is represented by

$$\frac{1}{K}\sum_{k=1}^{K} JT_i(k). \tag{2.9}$$

Consequently, the average speed of processor i, estimated by the fraction of the total load processed per ms, for the next iteration is approximately

$$\frac{1}{P\frac{1}{K}\sum_{k=1}^{K} JT_i(k)}. \tag{2.10}$$

Further, the expectation of the total processor speed of the P processors together is approximately the sum of those speeds:

$$\sum_{i=1}^{P} \frac{1}{P\frac{1}{K}\sum_{k=1}^{K} JT_i(k)}. \tag{2.11}$$

This implies that the expected time of the next iteration without send and rescheduling times given an optimal load distribution equals approximately

$$\frac{1}{\sum_{i=1}^{P} \frac{1}{P\frac{1}{K}\sum_{k=1}^{K} JT_i(k)}}. \tag{2.12}$$

Define for DLB iterations the optimal load fraction to be the proportion of the total load that a processor receives such that its job runtime equals the job runtimes of the other processors given that they all receive the optimal fraction of the load. As a consequence, the optimal load fraction of processor i is approximately

$$\frac{1}{\frac{1}{K}\sum_{k=1}^{K} JT_i(k)\sum_{i=1}^{P} \frac{1}{P\frac{1}{K}\sum_{k=1}^{K} JT_i(k)}} = \frac{P}{\sum_{k=1}^{K} JT_i(k)\sum_{i=1}^{P} \frac{1}{\sum_{k=1}^{K} JT_i(k)}}. \tag{2.13}$$

Finally, to simulate the $EJT_i(k)$ for all the iterations $k = 1, \ldots, K$, multiply the fractions with the real corresponding data values $JT_i(k)$ from the datasets.

$$EJT_i(k) = \frac{JT_i(k)P}{\sum_{k=1}^{K} JT_i(k)\sum_{i=1}^{P} \frac{1}{\sum_{k=1}^{K} JT_i(k)}}. \tag{2.14}$$

Step 3: Next, derive the $IT(k)$ for $k = 1, \ldots, K$ which is the maximum of the $EJT_i(k)$s of that iteration plus the $ST(k)$:

$$IT(k) = \max_{i=1,\ldots,P} EJT_i(k) + ST(k). \tag{2.15}$$

Step 4: Derive the running time of the SLB run by repeating step 3 K times, sum up all the $IT(k)$s, and add up one sample of the load rescheduling times:

$$S^*(t, P) := \sum_{k=1}^{K} IT(k) + RSchT(1), \tag{2.16}$$

Step 5: Derive the expected running time of an optimal SLB run on P processors by repeating steps 1 to 4 1000 times, and finally compute the average of the running times. As stated above, 1000 as the number of repetitions is high enough in order to derive stable and reproducible results.

2.7 Simulations of Dynamic Load Balancing

In this section, we describe the details of the trace-driven DLB simulations, and the computation of the run times of optimal load balancing. We use the definitions of 2.5.

2.7.1 Running times

The following steps have been incorporated in the simulations:

Step 1: Randomly select a resource set $S = \{p_1, \ldots, p_P\}$ of P processors from the datasets of the job runtimes, generated in Section 2.4, and order them such that the send times, measured in Section 2.4, between the processors are minimized. These numbers $p_1, \ldots, p_P$ correspond to the numbers of the datasets.

Step 2: The DES-based prediction $\hat{y}_i(k)$, $k = 1, \ldots, K$, (see for more details: Chapter 7) represents the predicted job-runtime on processor i. As a result, the expected speed of processor i (i.e., the fraction of the total load processed per ms) for the next iteration is

$$\frac{1}{P\hat{y}_i(k)}. \tag{2.17}$$

Further, the expectation of the total processor speed of the P processors together is

$$P^{-1} \sum_{i=1}^{P} \frac{1}{\hat{y}_i(k)}. \tag{2.18}$$

This implies, the expected time of the next iteration without send and rescheduling times given an optimal load distribution is

$$\frac{1}{P^{-1} \sum_{i=1}^{P} \frac{1}{\hat{y}_i(k)}}. \tag{2.19}$$

As a consequence, the optimal load fraction of processor i is

$$\frac{P}{\hat{y}_i(k)\sum_{i=1}^{P}\frac{1}{\hat{y}_i(k)}}. \tag{2.20}$$

Finally, to simulate the $EJT_i(k)$ of processor i for this and the next iterations $k = 1,\ldots,K$ before the next load rescheduling step, multiply the fractions with the real corresponding data values $JT_i(k)$ from the datasets.

$$EJT_i(k) = \frac{JT_i(k)P}{\hat{y}_i(k)\sum_{i=1}^{P}\frac{1}{\hat{y}_i(k)}}. \tag{2.21}$$

Step 3: Next, derive the $IT(k)$ for $k = 1,\ldots,K$ which is the maximum of the $EJT_i(k)$s of that iteration plus the $ST(k)$:

$$IT(k) = \max_{i=1,\ldots,P} EJT_i(k) + ST(k). \tag{2.22}$$

Step 4: Derive the running time of the DLB run, defined in Section 2.5 as $D(N,f,t,P)$, by repeating step 3 K times, sum up all the $IT(k)$s, and add up all the load rescheduling times:

$$D(N,f,t,P) := \sum_{k=1}^{K} IT(k) + \sum_{l=1}^{\lfloor K/N \rfloor} RSchT(l). \tag{2.23}$$

Step 5: Derive the expected running time of a DLB run on P processors by repeating steps 1 to 4 1000 times, and finally compute the average of the running times. As stated above, 1000 as the number of repetitions is high enough in order to derive stable and reproducible results.

2.7.2 Optimal Dynamic Load Balancing

In this section we compute the $D^*(t,P)$ which is the estimated running time with the optimal dynamic load balancing strategy, assuming all processor speeds are known in advance. The following steps have been incorporated in the computations:

Step 1: Randomly select a resource set $S = \{p_1,\ldots,p_P\}$ of P processors from the datasets of the job runtimes, generated in Section 2.4, and order them such that the send times, measured in Section 2.4, between the processors are minimized. These numbers $p_1,\ldots,p_P$ correspond to the numbers of the datasets.

Step 2: Assume that in an optimal DLB run that the predictions are perfect: $\hat{y}_i(k) = JT_i(k)$ for $k = 1,\ldots,K$. Consequently, the expected speed of processor i (i.e., the fraction of the total load processed per ms) for the next iteration is

$$\frac{1}{P\hat{y}_i(k)} = \frac{1}{PJT_i(k)}. \tag{2.24}$$

Further, the expectation of the total processor speed of the P processors together is

$$\sum_{i=1}^{P} \frac{1}{PJT_i(k)}. \tag{2.25}$$

Consequently, the optimal time of the iteration without send and rescheduling times given an optimal load distribution is

$$EJT_i(k) = \frac{P}{\sum_{i=1}^{P} \frac{1}{JT_i(k)}}. \tag{2.26}$$

Step 3: Next, derive the $IT(k)$ for $k = 1, \ldots, K$ which is the maximum of the $EJT_i(k)$s of that iteration plus the $ST(k)$:

$$IT(k) = \max_{i=1,\ldots,P} EJT_i(k) + ST(k). \tag{2.27}$$

Step 4: Derive the running time of the optimal DLB run by repeating step 3 K times, sum up all the $IT(k)$s, and add up all the load rescheduling times:

$$D^*(t, P) := \sum_{k=1}^{K} IT(k) + \sum_{l=1}^{\lfloor K/N \rfloor} RSchT(l). \tag{2.28}$$

Step 5: Derive the expected running time of an optimal DLB run on P processors by repeating steps 1 to 4 1000 times, and finally compute the average of the running times. As stated above, 1000 as the number of repetitions is high enough in order to derive stable and reproducible results.

2.8 Simulations of Job Replication

In this section, we describe the details of the trace-driven JR simulations, and the computation of the run times of optimal job replication. We use the definitions of 2.5.

2.8.1 Running times

Within a R-JR run, R replications of the same job are executed by a set of R different processors. For simplicity, we assume that P/R is an integer value. Consequently, the same groups of processors execute each iteration the same job. Thus, we are able to divide the P processors in P/R execution groups which all consist of R processors. We proceed along the following 6 steps to simulate the expected running time of a R-JR run on P processors.

Step 1: Randomly select a resource set $S = \{p_1, \ldots, p_P\}$ of P processors from the datasets of the job runtimes, generated in Section 2.4, and order them such that the send times, measured in Section 2.4, between the processors are minimized. These numbers $p_1, \ldots, p_P$ correspond to the numbers of the datasets.

Step 2: Divide the set of processors in execution groups. Execution group 1 consists of processors $p_1, \ldots, p_R$, group 2 consists of processors $p_{R+1}, \ldots, p_{2R}$, until group P/R that consists of processors $p_{P-R+1}, \ldots, p_P$. Define $EG(i)$ as the set of processors that are in the same execution group as processor i.

Step 3: In this step, derive the effective job-runtimes $(EJT_1(k), \ldots, EJT_P(k))$ for all P processors and iterations $k = 1, \ldots, K$. Therefore, derive first within each execution group which processor finished the same job as first. This can be done by taking one job-runtime value from each dataset of that execution group and observe which processor has the lowest job runtime. Within one execution group the $EJT(k)$ of the fastest processor e, $EJT_e(k)$, equals the job-runtime value $JT_e(k)$. The $EJT(k)$s of the other processors in the same execution group equal $EJT_e(k)$ plus the time that it takes to send the finalize-message from e to the other processors in the execution group, which is the above defined $FM_{e,j}(k)$:

$$EJT_i(k) = \begin{cases} JT_i(k) & \text{when } i = e, \\ \min_{j \in EG(i)} JT_j(k) + FM_{e,i}(k) & \text{else.} \end{cases} \tag{2.29}$$

Step 4: Next, derive the $IT(k)$ for $k = 1, \ldots, K$. This time can be derived by repeating step 2 R times (each processor gets R different jobs during one iteration), sum up the EJTs for each processor, taking the maximum of those sums, and adding up the $ST(k)$. Note that it is necessary to take into account all the previous $EJT(k)$s of each processor, because of the dependencies between consecutive $EJT(k)$s. We introduce parameter m, which indicate the step number within a iteration. The iteration time equals

$$IT(k) = \max_{i=1,\ldots,P} \sum_{m=1}^{R} EJT_i(k, m) + ST(k). \tag{2.30}$$

Step 5: Derive the running time of the R-JR run by repeating step 3 until all data values of the R datasets have been processed in the simulation and sum up all the ITs. In order to be able to compare the running times of runs with different values for R, multiply this sum with R to derive a comparable running time of K iterations for each possible R-JR run:

$$R(t, R, P) := R \times \sum_{k=1}^{\lfloor K/R \rfloor} IT(k). \tag{2.31}$$

When K/R is not an integer value, the value of the multiplying parameter, which is R in the above formula, for computing the running time of a R-JR run is $R\frac{Kmod(R)}{K-Kmod(R)}$.

Step 6: Derive the expected running time of a R-JR run on P processors by repeating steps 1 to 5 1000 times, and finally compute the average of the running times. As stated above, 1000 as the number of repetitions is high enough in order to derive stable and reproducible results.

2.8.2 Optimal Job Replication

Given P, the number of processors in the resource set, the running time of an optimal JR run is the lowest possible expected run time that can be obtained by JR. This is the run time that belongs to the JR run with the optimal number of replications, R^*. Compute the running time of the optimal JR run as follows.

$$R^*(t, P) = \min_{R=1,2,4,\ldots,P} R(t, R, P). \tag{2.32}$$

2.9 Implementation of Dynamic Load Balancing

In DLB schemes from time to time decisions are made to update the balancing of the loads on the basis of predictions of the processing speeds. To investigate whether DLB based on predictor f is an effective means to react on fluctuations in load or performance of processors we have implemented it in a representative SPMD program, called SOR (see Section 2.3 for more details). The job sizes (in SOR the number of rows) are adjusted such that they would take 2500 ms on a completely available 500 megahertz processor. Our implementation of the load balancing step is as follows. At the end of each iteration the processors predict their processing speed for the next iteration. After every N iterations the processors send their prediction to processor 0, the DLB scheduler. Subsequently, this processor calculates the "optimal" load distribution given those predictions, which is defined in (2.20), and sends relevant information to each processor. The load distribution is optimal when all processors finish their calculation exactly at the same time. Therefore, it is "optimal" when the number of rows assigned to each processor is proportional to its predicted processor speed. Finally, all processors redistribute the rows. In most of the experiments performed in this thesis, the total load balancing step takes around one third of the total time of one iteration (i.e., calculation and sending time).

3

STATISTICAL PROPERTIES OF JOB RUNTIMES

3.1 Introduction

As explained in Chapter 1, the main focus of this thesis is on highly CPU-intensive grid applications that require huge amounts of processor power for running tasks. Motivated by this, we performed in Chapter 2 measurements in a real global-scale grid environment in order to study the statistical properties of job runtimes on processors. Understanding the characteristics of a grid is also extremely useful for performing simulations or computations to assess the effectiveness of control strategies *prior* to their creation. For example, an effective means to do so is to implement dynamic load balancing (DLB) schemes that can dynamically update the load offered to different nodes in a grid in response to changing circumstances. The efficiency of such control schemes strongly depends on the effectiveness of prediction schemes, which in turn requires an understanding of the statistical properties of the dynamics of a grid environment.

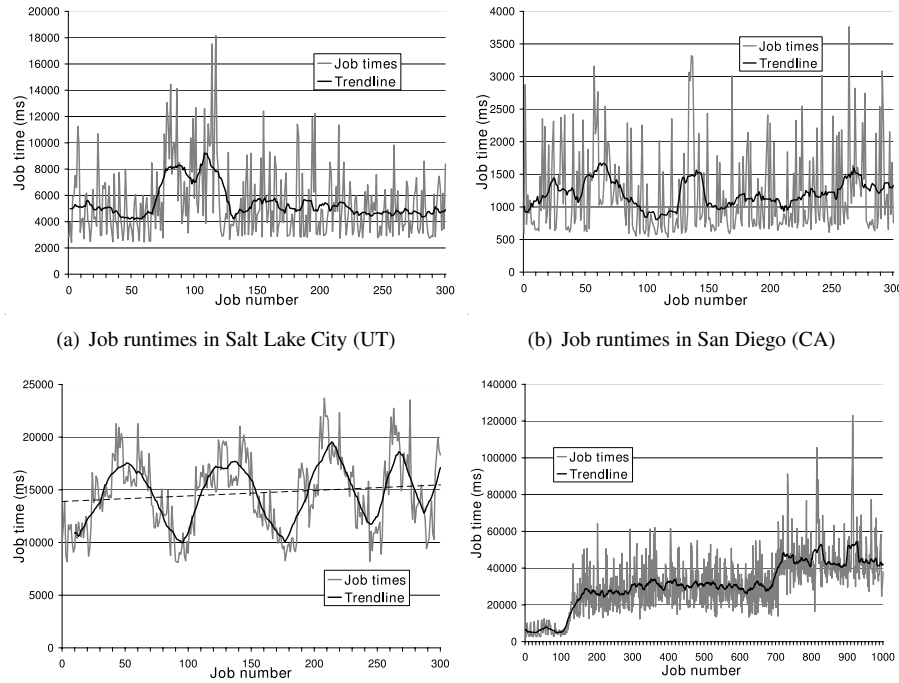
To this end, in this chapter* we investigate the statistical properties of job runtimes, which have been gathered in Section 2.4. The job-runtime measurements are multivariate time series, which means that the heights of the runtimes depend on many different variates (e.g., load). However, it is difficult to find relations between those variates and the job runtimes. Load is assumed to be the factor that has the most impact on job runtimes. However, as stated in Section 1.4.1 it is in practice hard to quantify its relation with job runtimes. Therefore, in this chapter we do not investigate the causes of the job-runtime fluctuations, and we are more interested in their properties. That corresponds to the idea that it will be hard to retrieve causes of fluctuations in the future grid.

The following steps are taken in this chapter. First we analyze plots of job runtimes, and make Box-and-Wisker plots to investigate the general characteristics of the datasets. Furthermore, in Section 3.4, we analyze the shape of the job-runtime distributions on the basis of histograms, and investigate the key factors on this shape. Subsequently, the Auto Correlation Function (ACF) and the Hurst parameter are inves-

*This chapter is based on papers [28] and [30].

tigated in respectively 3.5 and 3.6. In addition, in Section 3.7, we analyze the datasets with 17 other statistics. Next, in Section 3.8 we consider the correlation coefficients between all those different statistics. Finally, concluding remarks are given in Section 3.9.

3.2 Graphs



(a) Job runtimes in Salt Lake City (UT)

(b) Job runtimes in San Diego (CA)

(c) Job runtimes in Washington (DC)

(d) Long- and short-term job-runtime fluctuations in Tucson (AR)

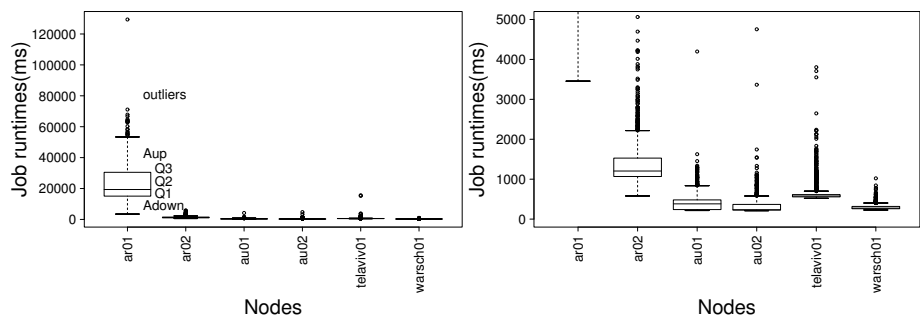
Figure 3.1: Job runtimes on 4 different nodes

In this section, the job runtimes, which have been collected in Section 2.4, are analyzed on the basis of plots. To this end, we selected parts (i.e., 300 or 1000 successive data-values) of four representative datasets which were generated on the following different sites: Salt Lake City (UT), San Diego (CA), Washington (DC), and Tucson (AR). Figures 3.1(a) to 3.1(d) depict the job runtimes on those sites plotted against the job numbers. The results lead to a number of interesting observations. We observe a strong heterogeneity between the different patterns with significant differences in the burstiness on the short time scale and the fluctuations on the longer time scales. For example, the short-term behavior in Figures 3.1(a) and 3.1(b) is much more bursty than the short-term behavior in Figure 3.1(c). Generally, the fluctuations at a short time scale take a few iterations (typically in the range 1 to 4 iterations), which roughly

corresponds to several minutes. Those short-term fluctuations are bursty and rather unpredictable. The fluctuations at an even longer time scale take several hundreds of iterations which corresponds to several hours in the time domain. These fluctuations are presumably caused by a changing load at the processor. Furthermore, for the longer-term fluctuations we observe a heterogeneity of patterns, including periodically changing behavior (see Figure 3.1(c)) and randomly changing behavior (see Figures 3.1(a), 3.1(b) and 3.1(d)). These observations also suggest that there is a great potential reduction in running times, which can be achieved by smart strategies that take into account those fluctuations in job runtimes.

3.3 Box-and-Whisker plots

Constructing Box-and-Whiskerplots, commonly known as Boxplots (see Chapter 2C in [68]), is a common way to represent the differences between the medians, the quartiles, the minimum, the maximum and the outliers of different datasets. A disadvantage of a representation of job runtimes by a Boxplot is that it does not depict the time dependencies. Figure 3.2 depicts the Boxplots of the running times (in milliseconds) of the datasets. The left-hand side plot gives a macroscopic view of the data, and the plot at the right-hand side focuses on running times ranging from 0 to 5000 ms. More precisely, we consider the three quartiles: the 25%-percentile (denoted as $Q1$), the median (denoted as $Q2$) and the 75%-percentile, $Q3$. These quartiles are plotted as a long horizontal line. In addition, we consider the statistical measures $A_{down} := Q1 - 1.5(Q3-Q1)$, $A_{up} := Q3+1.5(Q3-Q1)$, which are indications of the data points that should not be considered as outliers, and indicated by short horizontal lines. Finally, the outliers, which are defined by the Boxplots as points outside the range of A_{down} and A_{up}, are plotted by small circles. Because of this definition, it is possible that a Boxplot indicates a relatively large set of datapoints as outliers. Figures 3.2(a) and 3.2(b) lead



(a) Boxplot of running times (b) Boxplot of running times with y-range 0...5000

Figure 3.2: Boxplots of the 6 analysis phase datasets

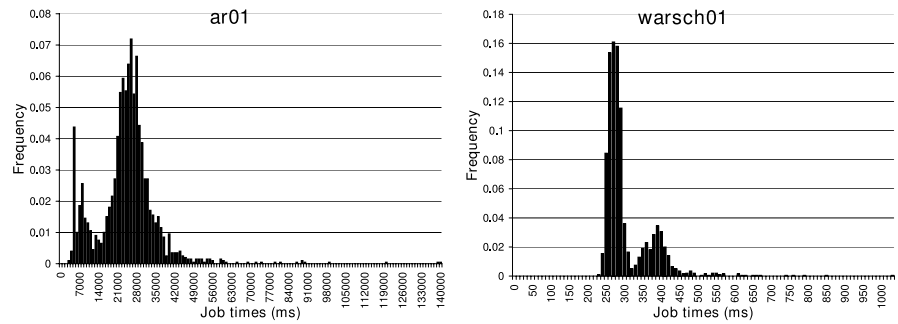
to the following observations: (1) the characteristics of the running times at a given node on different days in some cases differ strongly (see for example the results for ar01 and ar02), but can be quite similar in other cases (see for example au01 and au02), (2) the running-time characteristics of different nodes are strongly different, even when

experiments are done at the same time, (3) the running times at a given node within a given run are highly bursty, and have a large number of strong outliers, and (4) in most cases the outliers correspond to very large values of the running times, but in some cases (see for example ar01) outliers correspond to very small running times.

These observations address the need for prediction methods that can deal with heterogeneous and dynamically changing time series. Moreover, we reemphasize that the observed heterogeneity of characteristics of the running times in the grid environment differs *fundamentally* from the running-time characteristics in clusters of processors, which are usually homogeneous and well predictable.

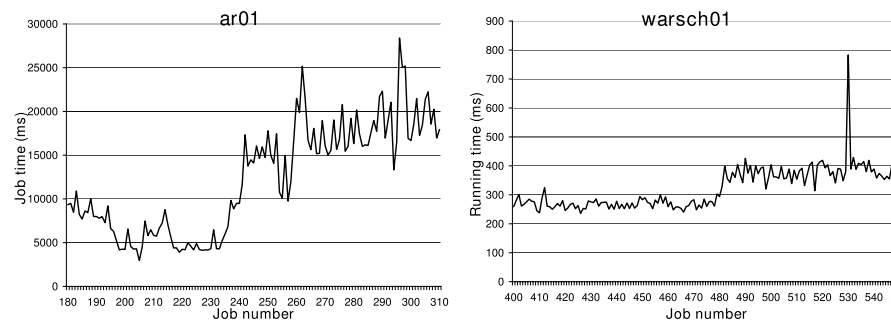
3.4 Histograms

To analyze the frequency distribution of the running times in more detail, Figure 3.3 shows the histograms of the marginal running-time distributions of two representative datasets. A disadvantage of a representation of job runtimes by a histogram is that it does not illustrate the time dependencies.



(a) Histogram of ar01 (b) Histogram of warsch01

Figure 3.3: Histograms of datasets ar01 and warsch01



(a) Level switch in ar01 (b) Level switch in warsch01

Figure 3.4: Level switches in datasets ar01 and warsch01

The results in Figure 3.3 show that the running-time distributions are typically multi-modal. This is caused by level switches in the running times over sustained time periods (ranging from minutes to hours). To illustrate this, Figure 3.4 gives a graphical representation of parts of datasets ar01 and warsch01. Figure 3.4(a) shows that from data points 235 to 260 a level switch occurs. From job number 180 to 230, the running times are between 3500 and 6000, which explains the first top in Figure 3.3(a). From job number 260 to 310 the measured running times are between 15000 and 30000, which explains the second top in Figure 3.3(a). Similar level switches are observed in Figure 3.4(b), which explains the bi-modal distribution of the running times in Figure 3.3(b). These level switches are presumably caused by changes in the processor load due to the launching or termination of other jobs on the same node. As stated by Dinda et al. [21, 23] much of the behavior of running times can be explained by changes in the load. However, they conclude that there are many other factors (e.g. memory space) that have their influence on the running times. In this thesis, we focus on reacting to the resulting fluctuations and not on prevention of the causes of fluctuations.

3.5 Auto Correlation Function

The Auto Correlation Function (ACF) is a way to investigate whether the data points show correlations over different time scales. For a given stationary set of data points $\underline{y} = (y_1, \ldots, y_N)$, an estimator for the ACF can be computed as follows:
for lags $h = 1, \ldots, N$,

$$\rho(h) \quad := \quad \frac{\gamma(h)}{\gamma(0)}, \tag{3.1}$$

with

$$\gamma(h) \quad := \quad \frac{1}{N} \sum_{t=1}^{N-h} (y_{t+h} - \mu(\underline{y}))(y_t - \mu(\underline{y})), \tag{3.2}$$

where h is the lag, and where $\mu(\underline{y}) := \sum_{t=1}^{N} y_t/N$.

Estimated ACFs of the job runtimes on ar01, au01, and telaviv01 are illustrated by Figure 3.5. Figure 3.5(e) depicts the $\ln$ of the ACF and its linear trend-line instead of the ACF. These ACFs are representative for all the datasets. The results show that the ACFs do not follow the same patterns; there are strong differences over the small (ranging from 1 up to 20) and large (higher than 20) lag values. For example, the ACF has significant autocorrelations over the small lag values of datasets ar01 and au01. On the contrary, the ACF for dataset telaviv01 decreases very quickly to 0, even for small lag numbers. For the datasets with significant autocorrelations for the large lag values, we consistently observe exponentially decaying autocorrelations, as illustrated by Figure 3.5(d), which suggests that the successive running times are short-range dependent. In Figure 3.5(e) this exponential relation can be seen more clearly, because the $\ln$ of the ACF has a linear relation with the lag. Periodicity can be observed by ACFs if one or more lag values are higher than the other lag values. Although some parts of the datasets temporarily show periodicity, we conclude that in general

the datasets do not show periodicity. However, we notice that the datasets represent job runtimes measured on one node with a time period of less than one day. For that reason, it is possible that the job runtimes show periodicity on longer times scales. We expect that if there exists a daily periodic behavior in the job runtimes of Planetlab nodes, this is not a strong behavior due to the fact that researchers at all time zones use the nodes.

3.6 Hurst parameter

Long-Range Dependence (LRD) is a statistical phenomenon which has received much attention in the field of telecommunications in the last ten years [11]. A time series is said to be LRD if it has correlations which persist over all time scales. LRD is characterized by the parameter H, the Hurst parameter, where $H \in (1/2, 1)$ indicates the presence of LRD. A Hurst parameter H of 1/2 indicates Short-Range Dependence (SRD). For a given set of data points $\underline{y} = (y_1, \ldots, y_N)$, let $\rho(h)$ be the auto-correlation function (ACF) of $\underline{y} = (y_1, \ldots, y_N)$, as defined in (3.1) and (3.2).

Definition 3.6.1: A stationary time series represented by $\underline{y} = (y_1, \ldots, y_N)$ with ACF $\rho(h)$ is said to be *long-range dependent* if the sum of estimated autocorrelations, $\sum_{h=-\infty}^{\infty} \rho(h)$, diverges.

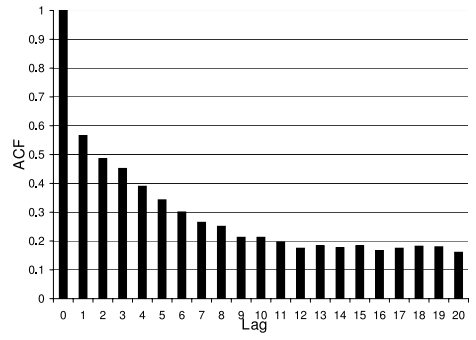
Often the specific functional form

$$\rho(h) \sim C_\rho h^{-\alpha}, \tag{3.3}$$

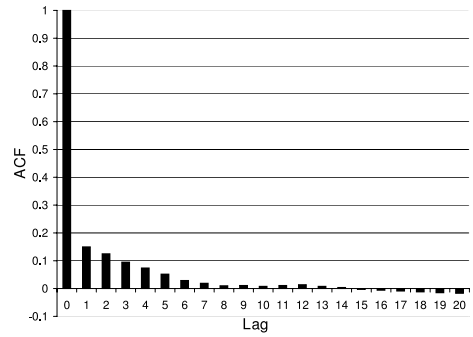
is assumed where $C_\rho > 0$ and $\alpha \in (0, 1)$. The parameter α is related to the Hurst parameter via the equation $\alpha = 2 - 2H$.

One method to estimate the Hurst parameter is to take the logarithm of the ACF, which is defined above, and estimate the gradient by a least-squares fit [75], which equals $-\alpha$. Next, derive H by the above formula. We estimate by this method the Hurst parameters of all 130 datasets, described in Chapter 2, to investigate if job runtimes on shared processors are SRD or LRD. Figure 3.6 depicts the histogram of the Hurst parameters of all the datasets. We observe that the Hurst parameter on average equals 0.73, and that a fraction of 0.63 of Hurst parameters is higher than 0.70.

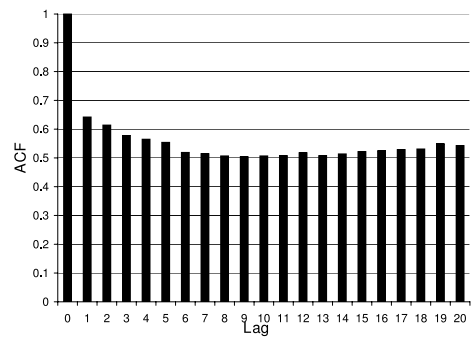
Figure 3.7 shows us how the estimated Hurst parameter changes over time. We estimated the Hurst parameter after each 100 measurements of job runtimes on a node in San Diego. The fluctuations of the Hurst parameter in this Figure is representative for all the nodes. Although a Hurst parameter determined from 100 measurements is not as accurate as when 2000 measurements are used, this figure depicts how the LRD changes fundamentally during runs.



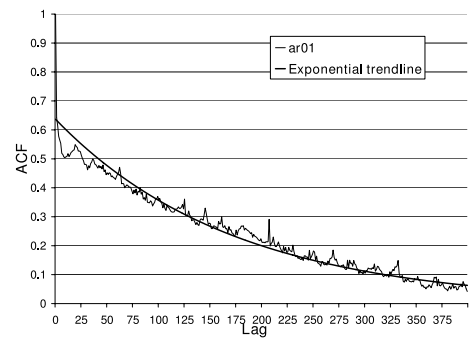
(a) au01



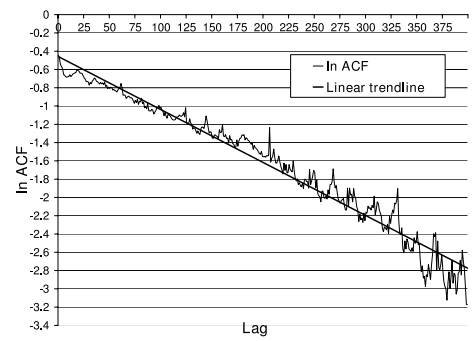
(b) telaviv01



(c) ar01



(d) ar01 (400 ACF values)



(e) ar01 (400 ln ACF values)

Figure 3.5: Plots of estimated ACFs for different datasets

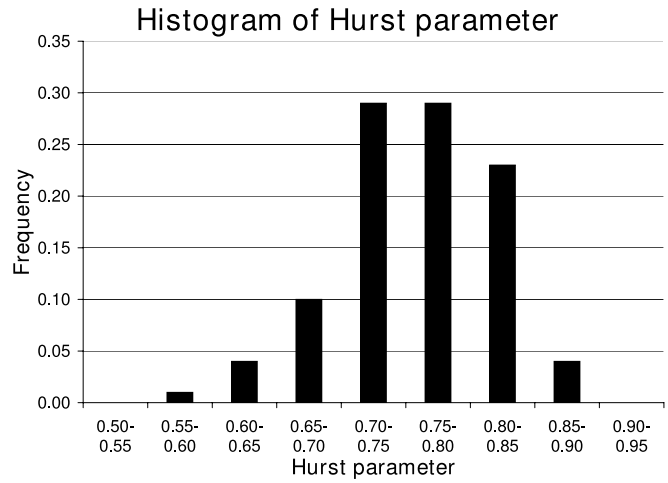


Figure 3.6: Histogram of Hurst parameters of 130 datasets

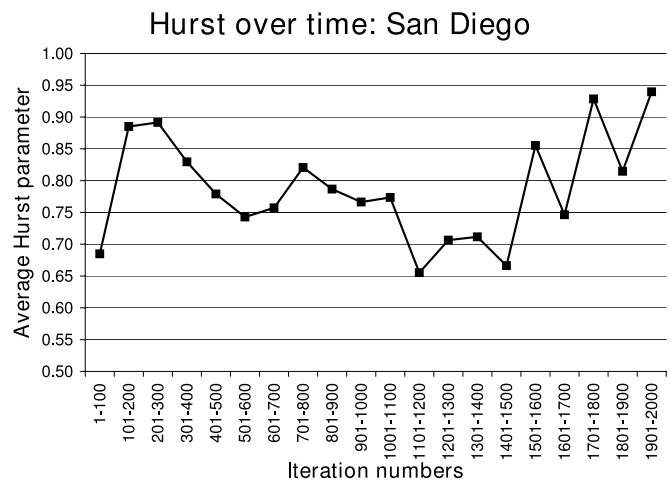


Figure 3.7: Estimated Hurst parameter over time on a node in San Diego

3.7 Other statistical properties

In this section, we provide an analysis of a new set of statistics that are relevant to investigating the statistical properties of job runtimes in grid environments. First, we define the long-term fluctuations or level switches as changes in the height of the data points that continue for more than 4 consecutive data points. To analyze the changes in the statistical properties over time, the index set $I := \{1, \ldots, N\}$ is partitioned into

$$I = I_1 \cup \cdots \cup I_M, \text{ where } I_k := \{B(k-1)+1, \ldots, Bk\} \text{ for } k = 1, \ldots, M,$$

where B is the block size and $M := N/B$ (assuming that M is integer-valued). Let 1_E be the indicator function of the event E, i.e., $1_E = 1$ if E is true and $1_E = 0$ otherwise. Further, let $\underline{y}_{I_k} := (y_{B(k-1)+1}, \ldots, y_{Bk})$, for $k = 1, \ldots, M$. For a vector $\underline{v}$, we define $|\underline{v}|$ to be the number of elements in vector $\underline{v}$, and therefore, $\underline{v} = (v_1, \ldots, v_{|\underline{v}|})$. Moreover, we define the sample mean and standard deviation of this vector by:

$$\mu(\underline{v}) := \frac{1}{|\underline{v}|} \sum_{t=1}^{|\underline{v}|} v_t, \tag{3.4}$$

and

$$\sigma(\underline{v}) := \sqrt{\frac{1}{|\underline{v}|-1} \sum_{t=1}^{|\underline{v}|} (v_t - \mu(\underline{v}))^2}. \tag{3.5}$$

Using this notation, we consider the set of statistical properties defined in Tables 3.1 and 3.2 below. Interpretations of the statistics and their formulas are given further on in this section.

Tables 3.1 represents the names of 17 different statistics that can provide insight in the statistical properties of job runtimes. Table 3.2 describes the formulas of those statistics. We compute the statistics by those formulas for six representative datasets, and moreover, the average, the standard deviation, the minimum, and the maximum of the statistics of all the 130 datasets defined in Section 2.4. Table 3.3 represents the values of those statistics. Throughout the whole analysis, the number of data points is $N = 2000$ and the block size $B = 20$.

To highlight the most interesting statistical properties, Figure 3.7 provides a graphical representation of the results for a number of statistics. From each group of statistics, the most important statistic is chosen to make a graphical representation. The results presented in Table 3.3 and Figure 3.7 are discussed in detail below.

The first group of statistics μ, σ, and c indicate the main characteristics of the datasets: statistic μ is the average, statistic σ the standard deviation, and statistic c is the Coefficient of Variation. The c equals the standard deviation divided by the average, and is a scale-invariant indicator for the variability of the data points.
We conclude from Table 3.3 and Figure 3.7 that the average and the standard deviations of the running times of the jobs may differ strongly between the datasets. Datasets can show similarities when two runs are performed on the same node and on two consecutive days, which is illustrated by au01 and au02. Table 3.3 and Figure 3.8(a) show that the cs of the running times are fairly low, ranging between 0.22 and 1.81.

Name	Statistic
μ	Average of whole dataset
σ	Standard deviation of whole dataset
c	Coefficient of Variation
$rmse_{lv}$	RMSE of last value
$\frac{\sigma}{rmse_{lv}}$	$\frac{\text{Standard deviation}}{\text{RMSE of last value}}$
$\sigma(\underline{\mu})$	Standard deviation of the averages
$c(\underline{\mu})$	Coefficient of Variation of the averages
$\frac{\sigma(\underline{\mu})}{\sigma}$	$\frac{\text{Standard deviation of averages}}{\text{Standard deviation}}$
$\sigma(\underline{\sigma})$	Standard deviation of standard deviations
$\frac{\sigma(\underline{\sigma})}{\mu}$	$\frac{\text{Standard deviation of standard deviations}}{\text{Average}}$
$c(\underline{\sigma})$	Coefficient of Variation of the standard deviations
f_{jumps}	Fraction of jumps7
σ_{jumps}	Standard deviation impact of jumps
$rmse_{jumps}$	RMSE impact of jumps
$\frac{\sigma_{jumps}}{\sigma}$	$\frac{\text{Standard deviation impact of jumps}}{\text{Standard deviation}}$
$\frac{rmse_{jumps}}{rmse_{lv}}$	$\frac{\text{RMSE impact of jumps}}{\text{RMSE of last value}}$
$\frac{mse_{jumps}}{\sigma^2}$	$\frac{\text{MSE impact of jumps}}{\text{variance}}$

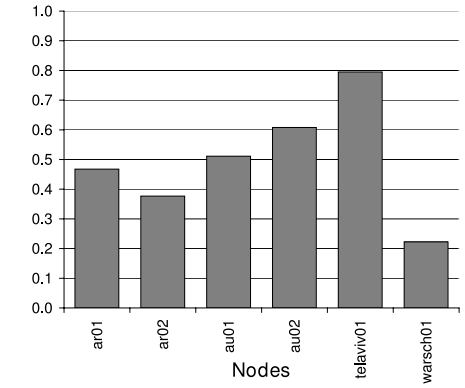
Table 3.1: Names of the statistics of the datasets

Name	Formula
μ	$\mu(\underline{y}) := \frac{1}{N}\sum_{t=1}^{N} y_t$
σ	$\sigma(\underline{y}) := \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(y_t - \mu(\underline{y}))^2}$
c	$c(\underline{y}) := \frac{\sigma(\underline{y})}{\mu(\underline{y})}$
$rmse_{lv}$	$rmse_{lv}(\underline{y}) := \sqrt{\frac{1}{N-1}\sum_{t=2}^{N}(y_t - y_{t-1})^2}$
$\frac{\sigma}{rmse_{lv}}$	$\frac{\sigma(\underline{y})}{rmse_{lv}(\underline{y})} := \frac{\sigma(\underline{y})}{rmse_{lv}(\underline{y})}$
$\sigma(\underline{\mu})$	$\sigma(\mu(\underline{y_{I_1}}), \ldots, \mu(\underline{y_{I_M}})) := \sqrt{\frac{1}{M}\sum_{k=1}^{M}(\mu(\underline{y_{I_k}}) - \mu(\underline{y}))^2}$
c($\underline{\mu}$)	$\mathrm{c}(\mu(\underline{y_{I_1}}), \ldots, \mu(\underline{y_{I_M}})) := \frac{\sqrt{B}\sigma(\mu(\underline{y_{I_1}}),\ldots,\mu(\underline{y_{I_M}}))}{\mu(\underline{y})}$
$\frac{\sigma(\underline{\mu})}{\sigma}$	$S_8(\underline{y}) := \frac{\sqrt{B}\sigma(\mu(\underline{y_{I_1}}),\ldots,\mu(\underline{y_{I_M}}))}{\sigma(\underline{y})}$
$\sigma(\underline{\sigma})$	$\sigma(\sigma(\underline{y_{I_1}}), \ldots, \sigma(\underline{y_{I_M}})) := \sqrt{\frac{1}{M}\sum_{k=1}^{M}(\sigma(\underline{y_{I_k}}) - \mu(\sigma(\underline{y_{I_1}}), \ldots, \sigma(\underline{y_{I_M}})))^2}$
$\frac{\sigma(\underline{\sigma})}{\mu}$	$S_{10}(\underline{y}) := \frac{\sigma(\sigma(\underline{y_{I_1}}),\ldots,\sigma(\underline{y_{I_M}}))}{\mu(\underline{y})}$
c($\underline{\sigma}$)	$\mathrm{c}(\sigma(\underline{y_{I_1}}), \ldots, \sigma(\underline{y_{I_M}})) := \frac{\sigma(\sigma(\underline{y_{I_1}}),\ldots,\sigma(\underline{y_{I_M}}))}{\mu(\sigma(\underline{y_{I_1}}),\ldots,\sigma(\underline{y_{I_M}}))}$
f_{jumps}	$f_{jumps}(\underline{y}) := \frac{1}{N-1}\sum_{t=2}^{N} 1_{\{\lvert y_t - y_{t-1}\rvert > 2\sigma(\underline{y})\}}$
σ_{jumps}	$\sigma_{jumps}(\underline{y}) := \sqrt{\frac{1}{N-1}\sum_{t=2}^{N} 1_{\{\lvert y_t - y_{t-1}\rvert > 2\sigma(\underline{y})\}}(y_t - \mu(\underline{y}))^2}$
$rmse_{jumps}$	$rmse_{jumps}(\underline{y}) := \sqrt{\frac{1}{N-1}\sum_{t=2}^{N} 1_{\{\lvert y_t - y_{t-1}\rvert > 2\sigma(\underline{y})\}}(y_t - y_{t-1})^2}$
$\frac{\sigma_{jumps}}{\sigma}$	$S_{15}(\underline{y}) := \frac{\sigma_{jumps}(\underline{y})}{\sigma(\underline{y})}$
$\frac{rmse_{jumps}}{rmse_{lv}}$	$S_{16}(\underline{y}) := \frac{rmse_{jumps}(\underline{y})}{rmse_{lv}(\underline{y})}$
$\frac{mse_{jumps}}{\sigma^2}$	$S_{17}(\underline{y}) := \frac{\sigma_{jumps}(\underline{y})^2}{\sigma(\underline{y})^2}$

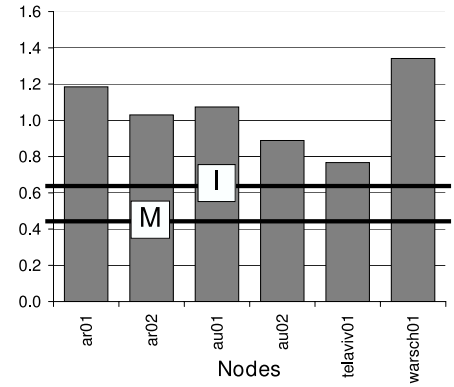
Table 3.2: Definitions of the statistics of the datasets

	Six representative datasets						All 130 datasets			
Name	**ar01**	**ar02**	**au01**	**au02**	**telaviv01**	**warsch01**	**average**	**standard deviation**	**minimum**	**maximum**
μ	23922	1346	407	301	656	300	3419	8449	107	42728
σ	11187	507	208	183	521	67	1817	4312	67	20560
c	0.47	0.38	0.51	0.61	0.79	0.22	0.57	0.26	0.22	1.81
$rmse_{lv}$	9443	493	194	206	680	50	1474	3224	50	14970
$\frac{\sigma}{rmse_{lv}}$	1.18	1.03	1.07	0.89	0.77	1.34	1.12	0.30	0.70	1.83
$\sigma(\underline{\mu})$	8471	303	123	92	166	48	1287	3392	29	16627
$c(\underline{\mu})$	1.57	1.03	1.34	1.39	1.11	0.72	1.36	0.57	0.63	3.62
$\frac{\sigma(\underline{\mu})}{\sigma}$	3.39	2.67	2.64	2.25	1.42	3.23	2.54	0.78	1.02	3.68
$\sigma(\underline{\sigma})$	4665	230	105	139	479	34	683	1466	23	6649
$\frac{\sigma(\underline{\sigma})}{\mu}$	0.20	0.17	0.26	0.46	0.73	0.11	0.24	0.25	0.08	1.73
$c(\underline{\sigma})$	0.80	0.65	0.76	1.64	2.64	0.98	0.71	0.63	0.18	3.56
f_{jumps}	0.03	0.03	0.02	0.04	0.01	0.03	0.06	0.06	0.01	0.22
σ_{jumps}	6265	330	108	148	488	35	794	1576	35	7096
$rmse_{jumps}$	8240	418	144	198	665	40	1107	2268	40	10499
$\frac{\sigma_{jumps}}{\sigma}$	0.56	0.65	0.52	0.81	0.94	0.53	0.53	0.17	0.24	0.96
$\frac{rmse_{jumps}}{rmse_{lv}}$	0.87	0.85	0.74	0.96	0.98	0.80	0.76	0.13	0.49	0.99
$\frac{mse_{jumps}}{\sigma^2}$	0.31	0.42	0.27	0.65	0.88	0.28	0.31	0.20	0.06	0.91

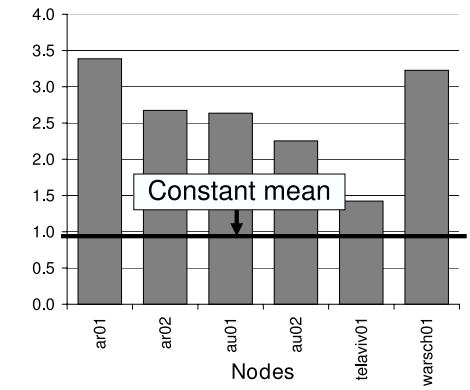
Table 3.3: Summary of the statistical properties of the datasets



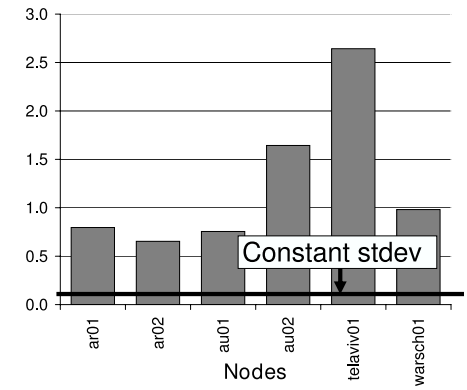
(a) Coefficient of Variation



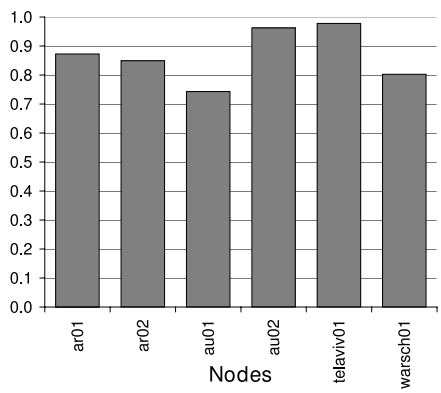
(b) Standard deviation over RMSE



(c) Standard deviation of the averages over the standard deviation



(d) Coefficient of Variation of Standard Deviations



(e) Fraction of total amount of fluctuations caused by jumps

Figure 3.8: Various statistical properties

Statistics $rmse_{lv}$ and $\sigma/rmse_{lv}$ indicate (1) to what extent the standard deviation is caused by short- or long-term fluctuations, and (2) how well the last value performs as a predictor for running times of jobs. Statistic $rmse_{lv}$ is the so-called Root of the Mean Squared Error (RMSE) of the last value, which is the RMSE of a predictor that uses the last value to predict the next value. Moreover, this statistic indicates the total amount of fluctuations. Statistic $\sigma/rmse_{lv}$ is the standard deviation over the RMSE of the last value, and is represented in Figure 3.8(b). Statistic $\sigma/rmse_{lv}$ theoretically varies between 0.5 and infinity. For (1), when statistic $\sigma/rmse_{lv}$ has the theoretical minimum value of 0.5 (see line M in Figure 3.8(b)), it indicates that the values alternate between two values; two random successive values show a correlation of -1. A value of $0.7(=\frac{1}{2}\sqrt{2})$ (see line I in Figure 3.8(b)) indicates that the values are independent and identically distributed. When the value for this statistical property is higher than 0.7 it indicates short- and long-term fluctuations; the higher the value the more long-term fluctuations influence the standard deviation. For (2), a value of 1.0 for this property indicates that the average and the last value as predictors have the same accuracy. The higher this value the better is the last value as predictor in comparison with the average. The results in Table 3.3 show a high variety of the values of statistic $rmse_{lv}$, ranging from 50 to almost 9500 for the six representative datasets and between 50 to almost 15000 for all the datasets. The average of statistic $\sigma/rmse_{lv}$ is 1.12, which is significantly higher than 0.7, which indicates long-term fluctuations. Moreover, it indicates that the last value is on average a slightly better predictor than the average. For example, for the two datasets au02 and tel01 that have a value lower than 1.0 an average predictor would predict better. We conclude from the high variety of the height of this statistic that the proportion long- and short-term fluctuations differs per dataset.

The next group statistics $\sigma(\underline{\mu})$, $c(\underline{\mu})$, and $\sigma(\underline{\mu})/\sigma$ indicate to what extent the average of the running times of jobs changes during the run. Statistic $\sigma(\underline{\mu})$ is the standard deviation of the averages, statistic $c(\underline{\mu})$ is the Coefficient of Variation of the averages, and statistic $\sigma(\underline{\mu})/\sigma$ is the standard deviation of the averages of the standard deviation. Statistics $c(\underline{\mu})$ and $\sigma(\underline{\mu})/\sigma$ are multiplied by a correction factor $\sqrt{B}$ to compensate the fact that a standard deviation of an average of B independent, identically distributed values is by definition $\sqrt{B}$ times smaller than the standard deviation of a single value. The closer the value of statistic $\sigma(\underline{\mu})$ is to 0, the more constant the average stays during the run. When statistic $c(\underline{\mu})$ has a value close to 1.0 or higher, the averages fluctuate significantly. A value of 1.0 for statistic $\sigma(\underline{\mu})/\sigma$ indicates that the expectation stays fairly constant during the run. Fluctuations in the standard deviations have no influence on this property.
Table 3.3 and Figure 3.8(c) show that for all datasets the averages fluctuate significantly: statistic $c(\underline{\mu})$ has an average value 1.36, and statistic $\sigma(\underline{\mu})/\sigma$ shows values that are significantly higher than 1.0. Nevertheless, the datasets show a high diversity in to what extent the averages fluctuate. We conclude from this group of statistics that all the datasets have many level switches.

Statistic $\sigma(\underline{\sigma})$, $\sigma(\underline{\sigma})/\mu$, and $c(\underline{\sigma})$ indicate how much the standard deviation of the running times changes during the run. Statistic $\sigma(\underline{\sigma})$ is computed by taking the standard deviation of standard deviations of B successive values. Statistic $\sigma(\underline{\sigma})/\mu$ equals statistic $\sigma(\underline{\sigma})$ divided by the average, and statistic $c(\underline{\sigma})$, which is graphically repre-

sented in Figure 3.8(d), equals statistic $\sigma(\underline{\sigma})$ divided by the average of the standard deviations. In general, the closer these statistics are to 0, the more the standard deviation stays constant during the run. If the running times are independent, identical normally distributed, statistic $c(\underline{\sigma})$ equals 0.16, which is independent of the height of the average and the standard deviation. We note that 0.16 depends on the value of B: the higher the B, the lower the value.
Table 3.3 and Figure 3.8(d) show that all the datasets have substantial fluctuations in the standard deviations. Nevertheless, there is diversity in the fluctuations of the standard deviations between the different sets. The running times on the node in Telaviv shows the most fluctuations in the standard deviation. For that node the high value for statistic $c(\underline{\sigma})$ is mainly caused by the many big outliers, as we see in Figure 3.2(b).

The last group of statistics f_{jumps}, σ_{jumps}, $rmse_{jumps}$, σ_{jumps}/σ, $rmse_{jumps}/rmse_{lv}$, and mse_{jumps}/σ^2 how many jumps the datasets contain and what the impact of those jumps is on the total amount of fluctuations, the standard deviation, and the variance. The first statistic of this group is the fraction of jumps, where a jump is defined as a value that differs (up- or downwards) from its previous value more than 2 times the standard deviation. A jump can be a peak or a level switch. The second statistic is the impact of the jumps on the standard deviation. This statistic sums up all the squares of the differences between the jumps and the averages, and takes the square root of that sum. The next statistic is the RMSE impact of jumps. This statistic sums up all the squares of the differences between the jumps and their previous values, and takes the square root of that sum. Statistic 15 indicates the fraction of the standard deviation that is caused by the jumps. It is computed by taking statistic σ_{jumps} over the standard deviation. Statistic $rmse_{jumps}/rmse_{lv}$ indicates the fraction of the total amount of fluctuations that is caused by the jumps. This property equals statistic $rmse_{jumps}$ divided by statistic $rmse_{lv}$. The last statistic, mse_{jumps}/σ^2, indicates how much influence the jumps have on the variance. This value equals the square of statistic σ_{jumps}/σ.
The results show that the fraction of the jumps does not differ that much between the datasets, and ranges mostly between 1% and the 22% with an average of 6%. The fraction of jumps is relatively low compared to the normal distribution (16%) and to the exponential distribution (13%). Statistics σ_{jumps} and $rmse_{jumps}$ illustrate huge differences between the different impacts of jumps on the standard deviation and on the variance. Statistics σ_{jumps}/σ and mse_{jumps}/σ^2 show that the fraction of the standard deviation and the variance that is caused by the jumps (respectively 53% and 31%) is slightly lower, but quite similar to the fractions of the normal distribution (respectively 60% and 36%) and the exponential distribution (respectively 73% and 53%). Moreover, the graphical representation of statistic $rmse_{jumps}/rmse_{lv}$, Figure 3.8(e), shows that on average 76% of the total amount of fluctuations (i.e. RMSE of last value) is caused by the jumps and that for two of the representative datasets the fraction is even higher than 95%. This value is significantly higher than those of the normal distribution (76%) and the exponential distribution (81%).

To summarize, the statistical data analysis of the datasets shows that: (1) there are significant differences between the characteristics of the datasets, (2) the datasets show on average more long-term than short-term fluctuations and the proportion differs per dataset, (3) the averages fluctuate significantly during the run, with differences in the

	μ	c	$rmse_{lv}$	$\frac{\sigma}{rmse_{lv}}$	$\sigma(\underline{\mu})$	$c(\underline{\mu})$	$\frac{\sigma(\underline{\mu})}{\sigma}$	$\sigma(\underline{\sigma})$	$\frac{\sigma(\underline{\sigma})}{\mu}$	$c(\underline{\sigma})$	f_{jumps}	σ_{jumps}	$rmse_{jumps}$	$\frac{\sigma_{jumps}}{\sigma}$	$\frac{rmse_{jumps}}{rmse_{lv}}$	$\frac{mse_{jumps}}{\sigma^2}$
μ	0.99	-0.08	0.98	0.24	1.00	0.22	0.38	0.97	-0.05	0.01	-0.16	0.95	0.96	-0.29	-0.06	-0.22
σ	1.00	-0.01	0.99	0.23	1.00	0.27	0.36	0.98	0.02	0.05	-0.15	0.96	0.97	-0.26	-0.04	-0.19
c		1.00	0.05	-0.35	-0.05	0.57	-0.43	0.11	0.84	0.54	0.14	0.14	0.09	0.52	0.44	0.49
$rmse_{lv}$			1.00	0.17	0.98	0.28	0.32	0.98	0.09	0.10	-0.13	0.97	0.99	-0.20	0.00	-0.13
$\frac{\sigma}{rmse_{lv}}$				1.00	0.25	0.34	0.86	0.22	-0.14	0.23	-0.70	0.19	0.15	-0.58	-0.36	-0.83
$\sigma(\underline{\mu})$					1.00	0.26	0.38	0.96	-0.02	0.02	-0.16	0.95	0.95	-0.29	-0.07	-0.22
$c(\underline{\mu})$						1.00	0.42	0.34	0.55	0.55	-0.40	0.34	0.30	0.06	0.15	-0.17
$\frac{\sigma(\underline{\mu})}{\sigma}$							1.00	0.34	-0.16	0.21	-0.74	0.29	0.30	-0.54	-0.36	-0.77
$\sigma(\underline{\sigma})$								1.00	0.20	0.21	-0.21	0.99	0.99	-0.16	0.05	-0.13
$\frac{\sigma(\underline{\sigma})}{\mu}$									1.00	0.88	-0.23	0.19	0.17	0.56	0.49	0.42
$c(\underline{\sigma})$										1.00	-0.51	0.19	0.16	0.43	0.43	0.15
f_{jumps}											1.00	-0.15	-0.14	0.31	0.26	0.61
σ_{jumps}												1.00	0.99	-0.13	0.04	-0.10
$rmse_{jumps}$													1.00	-0.14	0.07	-0.08
$\frac{\sigma_{jumps}}{\sigma}$														1.00	0.88	0.88
$\frac{rmse_{jumps}}{rmse_{lv}}$															1.00	0.71

Table 3.4: Correlation coefficients between the statistical properties

amount of fluctuations between the different nodes, (4) the standard deviations fluctuate significantly during the run, with differences in the amount of fluctuations between the different nodes, and (5) the datasets contain a small amount of jumps that have a huge influence on the standard deviation, the total amount of fluctuations, and the variance.

3.8 Correlation coefficients

In this section, we investigate to what extent the various statistical properties of the previous section are related with each other. Table 3.4 depicts the correlations between the statistical properties, which are defined in Table 3.1.

We make the following interesting observations from Table 3.4. First, the averages (the μs) and the standard deviations (the σs) of the datasets have a correlation coefficient of 0.99. This value for the coefficient means that there exists a strong linear relation between the average and the standard deviation of a dataset. Consequently, this illustrates that the datasets are scalable. Second, the fraction of jumps has a considerable negative correlation of -0.70 with statistics $\sigma/rmse_{lv}$, This value indicates that when the number of jumps increases (decreases), the number of long- as short-term fluctuations ($\sigma/rmse_{lv}$) decreases (increases), Third, the fraction of jumps has a correlation coefficient of -0.74 with $\sigma(\underline{\mu})/\sigma$. This correlation coefficient demonstrates that if the averages fluctuate more, the number of jumps decreases. Fourth, the number of jumps has a significant positive correlation of 0.61 with mse_{jumps}/σ^2. This demonstrates the linear relation between the fraction of jumps and the total impact of these jumps compared to the variance. Fifth, the mse_{jumps}/σ^2, and the $\sigma(\underline{\mu})/\sigma$ have a negative correlation coefficient of -0.77. This indicates that if there are more long-

term fluctuations in the job runtimes, the relative impact of the jumps on the variance is lower. Hence, the short-term fluctuations have a higher impact on the variance. Furthermore, we observe that many other statistical properties have a high correlation with each other. Nevertheless, those strong relations are related to the strong correlation between the averages and the standard deviations.

3.9 Conclusions

In this chapter, the statistical properties of the running times of tasks on processors and the relations between the statistics have been studied extensively. These investigations can be used to develop effective techniques for the prediction of running times. Moreover, they can be used to develop effective control schemes for robust grid applications.

We made a significant number of 18 observations. From the analyses with the Boxplots we observe (1) that the job runtimes measured on one node over different time scales mostly show a strong burstiness, (2) a strong heterogeneity of the characteristics of job running times among different hosts, even when the job runtimes are measured at the same time, (3) occurrence of sudden level-switches in the running times, and (4) that in most cases the outliers correspond to very large values of the running times, but in some cases outliers correspond to very small running times. Furthermore, the histograms show that the (5) job runtimes can not be fitted in a standard distribution. Moreover, the Auto Correlation Function analyses illustrate that (6) the characteristics of the Auto Correlation Functions of the different datasets differ completely, and (7) most datasets show exponentially decaying autocorrelations which indicates short-range dependency (SRD). In addition, the Hurst-parameter estimations indicate that (8) the job runtimes mostly show long-range dependence (LRD). Subsequently, we notice from the data analysis on the basis of an elaborate set of statistics that: (9) the characteristics of the job runtimes differ strongly per dataset, (10) the datasets show on average more long-term than short-term fluctuations and the proportion differs per dataset, (11) the averages and (12) the standard deviations fluctuate significantly during the run, with differences in the amount of fluctuations between the different nodes, and (13) the datasets contain a small amount of jumps that have a huge influence on the standard deviation, the total amount of fluctuations, and the variance. Finally, the investigations to the correlation coefficients between the different analyzed statistics demonstrate that (14) there exists a strong linear relation between the average and the standard deviation of the job runtimes on a node, (15) the lower (higher) the number of jumps, the more (less) long-term fluctuations compared to short-term fluctuations, and (16) the more (less) fluctuations of the averages, (17) the higher (lower) the fraction of jumps, the greater (smaller) the total impact of these jumps in comparison with the variance, and (18) short-term fluctuations have a greater impact on the variance than long-term fluctuations.

Most of the observations are consistent and some of them are trivial. However, there is a difference in the observations whether the job runtimes are SRD or LRD: the results of the Auto Correlation Function (ACF) analyses indicate that the job runtimes are SRD, and the Hurst-parameter estimations show that the job runtimes are LRD. Based on the analyses, we expect that there indeed exists long-range dependency among the job runtimes. The estimations of the ACFs need a higher number of measurements

than the Hurst-parameter analyses in order to conclude whether a time series is LRD. We presume that 2000 as the number of data points was not enough for the ACF to affirm LRD. However, more analyses are needed to confirm the expectation that the job runtimes are LRD.

To summarize, our experiments show strong heterogeneity in the characteristics of the job runtimes on different globally-distributed shared processors.

4

PERFORMANCE ASSESSMENT OF LOAD BALANCING STRATEGIES

4.1 Introduction

As pointed out in Chapter 1, it is essential to develop effective means to cope with the fluctuations in grid environments. The principles of Static and Dynamic Load balancing (respectively SLB, and DLB) are means to cope with fluctuating processing speeds in a heterogeneous environment. In Chapter 2, the grid testbed Planetlab, and the procedure to extensively collect data in this testbed have been described. Moreover, simulation and implementation details of the different load balancing methods, ELB, SLB, and DLB, have been given. In addition, the computations to derive the highest possible speedup are specified. In this chapter we investigate the effectiveness of those load balancing strategies in a global-scale grid environment on the basis of comprehensive trace-driven simulations and real implementation experiments. The experiments were performed with the Successive Over Relaxation (SOR) application, discussed in Chapter 2.

This chapter* is organized as follows. First in Section 4.2, we perform trace-driven simulations with Static Load Balancing (SLB). We investigate the impact of the number of "cold" iterations to estimate the average job runtimes on the speedup compared to Equal Load Balancing (ELB). Next, we investigate in Section 4.3 the consequences of the number of measurements between two load rescheduling steps on the running times of DLB. Subsequently, we analyze the effectiveness of DLB for different communication to computation ratios. Moreover, in Section 4.4 we verify the simulation results by experiments of real implementations of DLB in the global-scale grid testbed Planetlab. Different job sizes will be tested. Finally, in Section 4.5 we summarize the conclusions.

*This chapter is based on papers [25], [26], and [27].

4.2 Static Load Balancing

As shown in Chapter 1, Static Load Balancing (SLB) strategies use a number of "cold iterations" to estimate the average processor speeds, in order to balance the load. In this section, we simulate trace-driven different Static Load Balancing (SLB) settings in order to investigate the speedups of different numbers N of job-runtime measurements before the static load balancing step. To this end, we incorporate the datasets of the USA nodes Boston, Pasadena, Salt Lake City, San Diego, Tucson, and Washington in the simulations. For each simulation, we randomly select four processors from this set. We investigate the SLB possibilities of a parallel program with problem size $t = t_{default} = 2.5$, and which runs on $P = 4$ processors. We simulate the cases that $N = 0, 1, 10, 20, 50, 100, 200, \ldots, 1000$, and investigate the upper bound of the speedup for this choice of parameters. Table 4.1 depicts the average speedups with and without the overhead of the single load rescheduling step where each value is derived from 10 trace-driven simulation runs. The details of the SLB trace-driven simulations have been described in Section 2.6. The $S(N, t, P)$ and the $S^*(t, P)$ are defined in 2.5.

LB strategy	**Average speedup without overhead**	**Average speedup with overhead**
$S(0, 2.5, 4)$	1.00	1.00
$S(1, 2.5, 4)$	1.19	1.16
$S(10, 2.5, 4)$	1.18	1.15
$S(20, 2.5, 4)$	1.26	1.21
$S(50, 2.5, 4)$	1.31	1.25
$S(100, 2.5, 4)$	1.32	1.25
$S(200, 2.5, 4)$	1.31	1.25
$S(300, 2.5, 4)$	1.50	1.39
$S(400, 2.5, 4)$	1.60	1.45
$S(500, 2.5, 4)$	1.60	1.46
$S(600, 2.5, 4)$	1.58	1.44
$S(700, 2.5, 4)$	1.55	1.42
$S(800, 2.5, 4)$	1.49	1.38
$S(900, 2.5, 4)$	1.44	1.34
$S(1000, 2.5, 4)$	1.39	1.30
$S^*(2.5, 4)$	1.91	1.61

Table 4.1: Speedups of static load balancing strategies

We conclude the following from the results in the table. First, we notice from the analyses that include overheads that the highest SLB speedups can be gained when $N = 500$. For this case, on the one hand sufficiently many measurements have been applied in determining the average job runtimes such that accurate averages have been derived, and on the other hand, there are still enough remaining iterations to apply the new load distribution to and gain in running time. Second, we observe that unless during a complete run only one load rescheduling step is applied, the speedup is considerable. Third, the table points out that there is still a significant difference between

the highest derived SLB speedup and the optimal SLB strategy. Fourth, the decreasing impact on the speedup of the load-balancing step overhead is very low. This impact increases and becomes significant when the speedup is higher.

4.3 Dynamic Load Balancing

Number of iterations between Load Balancing steps

In this section, we investigate the optimal number N of iterations between two rescheduling steps in a DLB run. To this end, we incorporate the same datasets, as used above, of the USA nodes Boston, Pasadena, Salt Lake City, San Diego, Tucson, and Washington in the simulations. For each simulation, we randomly select four processors from this set. We investigate the cases of a parallel program with problem size $t = t_{default} = 2.5$, and which runs on $P = 4$ processors. We simulate the cases that $N = 0, 1, 2, 3, 4, 510, 20, 30, 40, 50, 100, 200, 300, 400, 500$, and investigate the maximal possible speedup for this choice of parameters.

Table 4.2 shows the speedups that can be made by dynamic load balancing on the basis of ES with $\alpha = 0.5$ predictions, compared to the case with equal load balancing, for a variety of load balancing strategies. Moreover, the table illustrates the impact of the rescheduling overhead on the speedup. Based on extensive experimentation with the value of α, we found that a suitable value of α is 0.5. The $D(N, f, t, P)$ and the $D^*(t, P)$ are defined in 2.5.

The results shown in Table 4.2 lead to a number of interesting observations. First, we notice that for the case without overhead N is 1 provides the highest speedup. This observation is expected, because with $N = 1$, the load is balanced every iteration and therefore always up-to-date. For the case without load-balancing overhead these updates cost no extra time. First, we observe that a suitable value of N is 10 with taking into account the overhead. Second, we observe that there is a high potential speedup by properly reacting to fluctuations of processing speeds by dynamic load balancing. The potential speedup is shown by the speedup of $D^*(2.5, 4)$ in Table 4.2; in the optimal dynamic load balancing case it is possible to obtain a speedup of 1.85 assuming that the overhead of the optimal DLB strategy approximately equals the overhead of the DLB strategy with N is 10. Third, we observe that despite the inaccuracy in the predictions of the calculation times the highest obtained speedup factor by applying DLB is still close to the *theoretical* optimum.

Communication to computation ratio

In this section, we analyze the running times of DLB for different communication to computation ratios by trace-driven simulations with extensive data sets, which are both described in Chapter 2. First, we divided the datasets of the 130 runs into two sets of datasets: one set contains 40 datasets and the other 90 datasets. In order to compare the results of Sections 4.2 and the above section with the simulation results of this section, the first set consists of datasets which are generated from the same nodes. The second set contains datasets generated from in total 22 different nodes which includes datasets generated from the same nodes which are used for set one. Set one only consists of nodes in the USA: Boston, Pasadena, Salt Lake City, San Diego, Tucson, and Wash-

LB strategy	Average speedup without overhead	Average speedup with overhead
$D(0, ES(0.5), 2.5, 4)$	1.00	1.00
$D(1, ES(0.5), 2.5, 4)$	2.52	1.57
$D(2, ES(0.5), 2.5, 4)$	2.48	1.71
$D(3, ES(0.5), 2.5, 4)$	2.45	1.76
$D(4, ES(0.5), 2.5, 4)$	2.42	1.78
$D(5, ES(0.5), 2.5, 4)$	2.40	1.79
$D(10, ES(0.5), 2.5, 4)$	2.34	1.80
$D(20, ES(0.5), 2.5, 4)$	2.28	1.80
$D(30, ES(0.5), 2.5, 4)$	2.28	1.80
$D(40, ES(0.5), 2.5, 4)$	2.25	1.79
$D(50, ES(0.5), 2.5, 4)$	2.24	1.79
$D(100, ES(0.5), 2.5, 4)$	2.18	1.76
$D(200, ES(0.5), 2.5, 4)$	2.02	1.68
$D(300, ES(0.5), 2.5, 4)$	2.05	1.69
$D(400, ES(0.5), 2.5, 4)$	1.82	1.56
$D(500, ES(0.5), 2.5, 4)$	1.74	1.52
$D^*(2.5, 4)$	2.50	1.85

Table 4.2: Speedups of Dynamic Load Balancing strategies

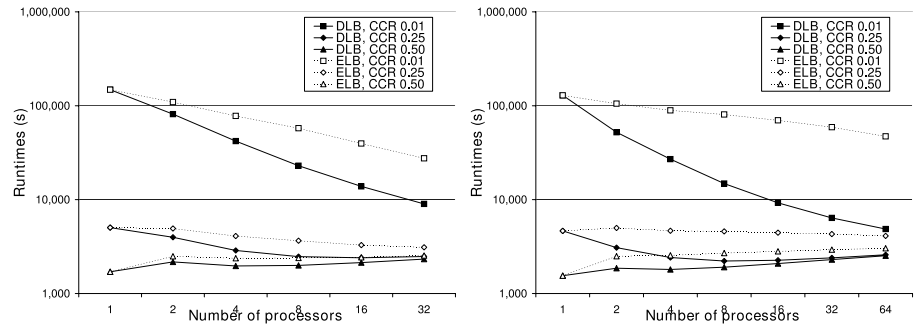
ington DC. Furthermore, as described in Chapter 2, the second set contains, besides the datasets which are generated on the same nodes as set one, datasets which are generated on the following different nodes: Amsterdam, The Netherlands; Cambridge, UK; Beijing, China; Copenhagen, Denmark; Le Chesnay, France; Madrid, Spain; Moscow, Russia; Santa Barbara, USA; Seoul, South Korea; Singapore; Sydney, Australia; Tel Aviv, Israel; Taipei, Taiwan (Academica Sinica); Taipei, Taiwan (National Taiwan University); Vancouver, Canada; and Warsaw, Poland.

The job runtimes in set one are on average approximately 72500 ms, and in set two 65000 ms. Further analysis shows that the job runtimes on the nodes in set two show more burstiness and have higher differences between the average job runtimes on the processors. That last property is mainly caused by the fact that the nodes in set two are globally distributed and the nodes in set one are distributed within the USA; set one shows more coherence between the generated datasets.

First, we simulate the running times of DLB for different numbers of processors with set one and two. Furthermore, we analyze the impact of the communication-to-computation rate (CCR) on the run times of DLB. The average CCR, as defined in Section 2.5, is found to be 0.01 in our datasets. In order to investigate this impact, we linearly interpolate the heights of the computation times such that we are able to derive simulations of runs with a CCR of 0.25, and of 0.50. With those interpolated job runtimes, we again simulate runs based on DLB for a wide range of situations.

Below, we present the results of the simulations of the DLB runs. We investigate the DLB running-times with both sets of processors for runs with a CCR of 0.01, 0.25, and 0.50 on 1, 2, 4, 8, 16, and 32 processors. Figure 4.1(a) depicts the average run times on a logarithmic scale of all performed simulations on nodes of set one. Moreover,

Figure 4.1(b) depicts the average run-times on a logarithmic scale of all the performed simulations on nodes of set two.



(a) 40 datasets of USA nodes

(b) 90 datasets of globally distributed nodes

Figure 4.1: DLB Run-times for different CCRs

From the simulation results of the runs with a CCR of 0.01, we conclude that selecting more processors in the run decreases the running times, which is the main motivation for programming in parallel. Although the rescheduling and send times increase when more processors are selected in the run, the decrease in the computation times for this case is always higher. As is shown by Figures 4.1(a) and 4.1(b), we draw different conclusions when the CCR is higher than 0.01. For runs on nodes of set one and a CCR of 0.25, we notice a decrease in running times until the amount of 16 processors is selected. When more processors have been selected, the running times will increase due to the significant heights of the rescheduling- and send times. Furthermore, we conclude that for every experimental setting, DLB consistently shows a speedup in comparison with ELB, even for runs with a CCR of 0.50. However, for the case that CCR equals 0.50, we observe that the number of processors with the lowest run times is one, which means that it is not efficient to run the program in parallel on more than one processors.

4.4 Experiments

In this section, we verify the simulation results of the section above by experiments of real implementations of DLB in the global-scale grid testbed Planetlab. We conduct experiments in two parts. In the first part, we perform extensive experiments with a parallel program that consists of jobs with the problem size $t = 2.5$ (defined in 2.5) and analyze the difference between the run times of ELB and DLB. In the second part, we perform experiments to investigate the dependence between the problem sizes and the speedup gained by implementing DLB. Therefore, we run the two versions of the SOR for job sizes with problem sizes $t = 1.25, 2.5, 3.75, 5.0$. Furthermore, by analyzing in more detail the obtained experimental results of all the experiments, we investigate how the load distribution changes during a DLB run. Finally, we analyze how run times can evolve over time.

Ideally, experiments should be performed both with a small number of nodes and with a very large number of nodes. To obtain statistically significant results, the experimental results need to be reproducible. In practice, however, the most commonly used grid testbeds are not yet mature enough, and the availability of many nodes is limited. For this reason, we choose to conduct our experiments with four sites. On the one hand, the number of four sites is large enough to demonstrate a significant speedup factor by DLB. On the other hand, this number is small enough to reproduce experiments with the same set of nodes within a reasonable time frame. We choose to only use nodes in the USA or Canada, because these nodes have the highest up times. We use two sets of four nodes of Planetlab to conduct our experiments. Set one consists of the nodes Pasadena (CA), Tucson (AR), Washington (DC) and Boston (MA), and set two consists of Vancouver (BC), San Diego (CA), Salt Lake City (UT) and Chicago (IL). The nodes are connected by the Internet via a linear structure, as shown by Figure 4.2.

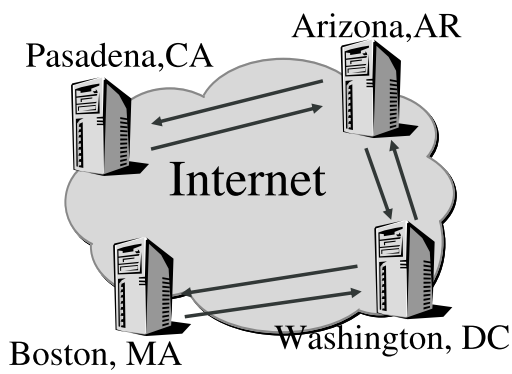


Figure 4.2: Nodes on Planetlab, used for experiments

We perform our experiments with the Successive Over Relaxation (SOR) application, as described in Section 2.3. The default load balancing scheme is referred to as Equal Load Balancing (ELB). ELB assumes no prior knowledge of processor speeds of the nodes, and consequently balances the load equally among the different nodes. Ideally, experiments with and without DLB should be performed simultaneously. Unfortunately, in practice performing experiments simultaneously is not possible because of interference. Therefore, to make a fair comparison we alternatingly run the two implementations under comparable circumstances: each day at 09:00 CET we start one of the two implementations, and the next day we start the other one.

In the first part of the experiments, we consider the effectiveness of implementing DLB based on ES. We have performed 30 runs with the original ELB SOR implementation and 30 runs of the DLB implementation with set one of Planetlab nodes. To obtain statistically relevant results it is necessary to perform as many as 30 runs. To make a fair comparison, the runs were alternatingly performed with ELB and DLB. The odd run numbers correspond to DLB-based experiments, and the even run numbers are based on ELB. One run consists of 2000 iterations. Interrupted runs are omitted. Figure 4.3 shows the running times for these experiments. The results plotted in Figure 4.3 show that the DLB-based experiments are significantly faster than their ELB-counterparts,

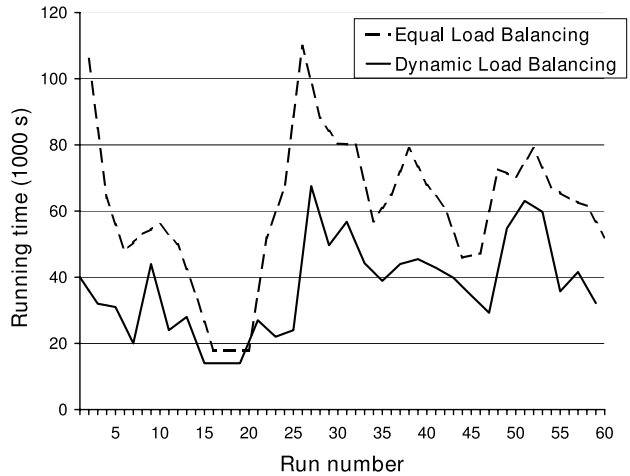


Figure 4.3: Running times DLB compared to ELB

consistently over all experiments (except for a single outlier in run 20). Interestingly, the DLB strongly outperforms ELB independent of the actual running times. The average speedup factor by using DLB instead of ELB was found to be roughly a factor of 1.8. This confirms the results of the simulations addressed above.

Another interesting question is how the running times achieved by implementing DLB depend on the problem size. Therefore, in the second part of experiments we have performed experiments with DLB and ELB and with different problem sizes $t = 1.25, 2.5, 3.75, 5.0$ on set two of Planetlab sites. Those runs consist of 1000, 500, 375, and 250 iterations, respectively. The experiments have been repeated seven times in order to obtain reliable estimates. Interrupted runs are omitted. Figure 4.4 shows the average running time as a function of t. This figure shows that the running time

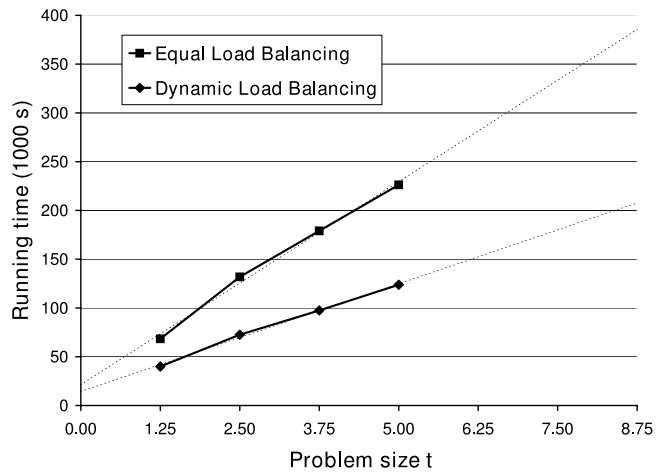


Figure 4.4: Running times as a function of the problem size t

Problem size t	CCR	Average speedup
0.05	*0.500*	*1.50*
0.10	*0.250*	*1.53*
1.25	0.020	1.71
2.50	0.010	1.82
3.75	0.007	1.82
5.00	0.005	1.83

Table 4.3: Speedup factors for different problem sizes

increases nearly linearly in the number of rows, for both ELB and DLB. More precisely, based on a simple least-square estimation method we obtain the following approximate expression for the running times $E(t,4)$ and $D(10, ES(0.5), t, 4)$ (in seconds) as a function of the problem size t:

$$E(t,4) = 41600t + 21138, \tag{4.1}$$

$$D(10, ES(0.5), t, 4) = 22200t + 14363. \tag{4.2}$$

The offset for ELB consists of send and wait times, which are independent of the problem size. The offset for DLB also consists of send times and wait times, but in the DLB case the wait times are smaller than in the ELB case, because DLB is able to react to temporary imbalance causing larger wait times. In addition, the DLB-offset contains the overhead involved performing load balancing actions. Table 4.3 shows the speedup factor for different values of t and their corresponding CCR. In order to compare the results with the simulations, we compute estimations with (4.1) and (4.2) for the speedups of DLB programs with a CCR of 0.50 and 0.25.

Table 4.3 demonstrates that, because of the above-mentioned differences in the offsets for ELB and DLB, the speedup depends on the problem size. More precisely, it follows directly from (4.1) and (4.2) that in the current experimental setting the speedup factor seem to converge to the following constant if t grows large:

$$\lim_{t\to\infty} \frac{E(t,4)}{D(10, ES(0.5), t, 4)} = \frac{41600}{22200} \approx 1.87.$$

Moreover, the estimated speedups of 1.50 with a CCR of 0.50 and 1.53 with a CCR of 0.25 are higher than the speedups generated in Section 4.3. The following two reasons are possible causes for this difference: (1) the sets of nodes are different, and (2) formulas (4.1) and (4.2) possibly lose accuracy for CCRs that differ significantly from the CCRs that are used to estimate the formulas.

To analyze the speedup between DLB and ELB in more detail, Figures 4.5 and 4.6 show the evolution of the load distribution over the different nodes for both the DLB and the ELB scheme. More precisely, a representative development of the load assigned to the nodes is shown. For the DLB case we observe both short- and long-term

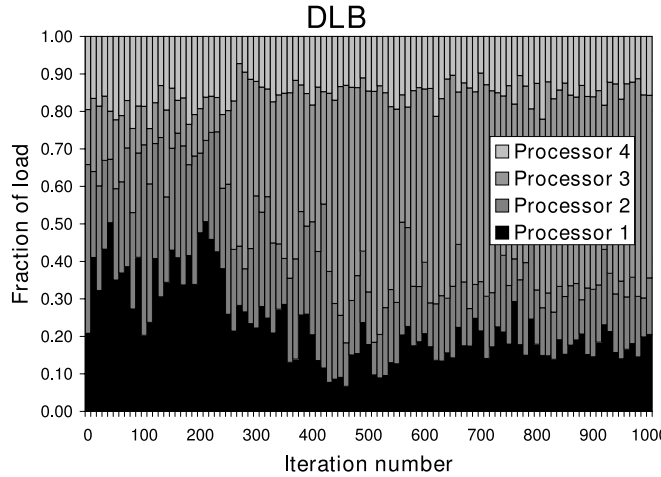


Figure 4.5: Fraction of load on each processor in a DLB run

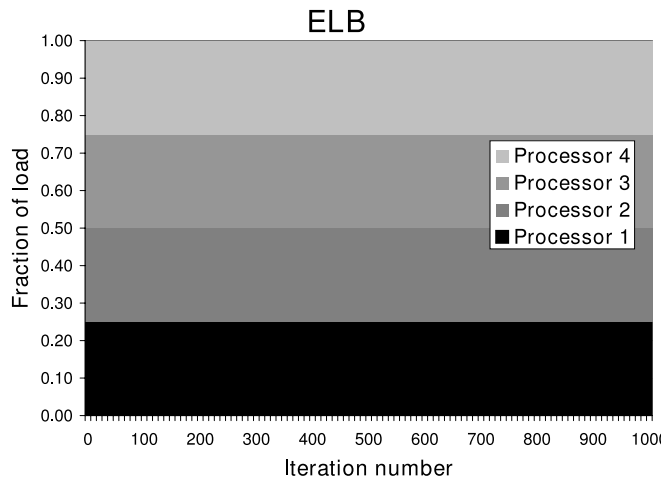


Figure 4.6: Fraction of load on each processor in a ELB run

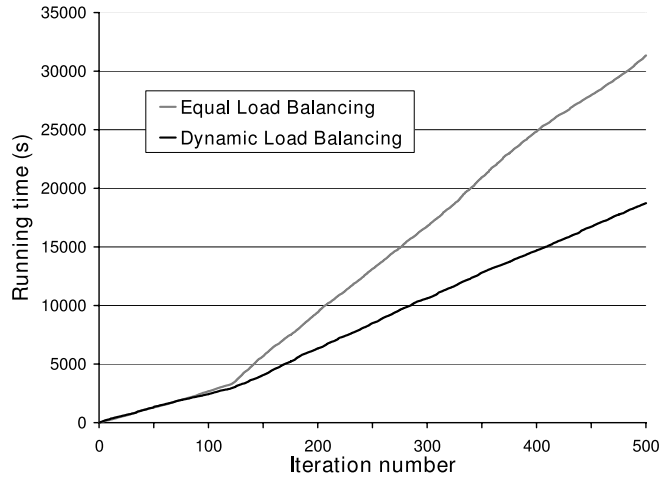


Figure 4.7: Cumulative running time as a function of the iteration number

changes in the number of rows during a run, which are caused by dynamic reactions on the short- and long-term changes in calculation times. The results also show the drawback of implementing Static Load Balancing (SLB) schemes, where the load is balanced statically on the basis of the first N iterations. A key problem is to find a suitable value for N, which is based on the following trade-off. If N is too large, then the benefit of SLB is marginal by definition. If N is too small, then the estimates of the processing speeds of the nodes, and hence of the "optimal" load distribution, are unreliable. For example, the results in Figure 4.5 show that the optimal load distribution on the basis of DLB is roughly $12\%, 18\%, 54\%$ and 16%, for processor 1 to 4, respectively. However, if SLB were used with $M \leq 250$ then the SLB weights would be roughly $20\%, 16\%, 34\%$ and 30%, respectively.

The next question is how the speedup based on DLB compared to ELB evolves over time. To this end, Figure 4.7 shows the cumulative running time as a function of the number of iterations for DLB and ELB, respectively, for the same experiment as in Figure 4.5. Figure 4.7 shows that at the beginning of the run DLB is not faster than the original implementation ELB. However, after about 120 iterations, the ELB run tends to slow down significantly, whereas the DLB slows down only marginally. This observation can be explained from Figure 4.5 as follows. During the first (say) 120 iterations, processors 1 and 2 were relatively fast compared to processors 3 and 4. However, around iteration 120 for some unknown reason processors 1, 2 and 4 were slowing down possibly caused by background load, whereas the processing speed of processor 3 did not change significantly. Consequently, the DLB scheme dynamically assigned additional rows to processor 3, while the static ELB scheme did not. In this way, the DLB scheme was found to properly react to changes in the effective processor speeds, and as such outperformed ELB significantly.

4.5 Conclusions

We conclude the following from the results in this chapter. First, we showed that SLB in a global-scale grid environment leads to speedups. We notice that the SLB speedups are as high as possible when $N = 500$. For this case, on the one hand enough measurements have been applied in determining the average job runtimes such that accurate averages have been derived, and on the other hand, there are still enough remaining iterations to apply the new load distribution to and gain in running time. However, the results point out that there is still a significant difference between the highest derived SLB speedup and the optimal SLB strategy. We expect that in an ever changing global-scale processor environment, like the grid, the difference between the speedups of SLB methods and their theoretical upper-bounds always remains due to the unpredictableness of the fluctuations in the job runtimes.

Second, in this chapter trace-driven simulations show that DLB leads to higher speedups than SLB, despite the fact that the load rescheduling phase increases the overhead. We conclude that load balancing every $N = 10$ iterations leads to the highest speedups in parallel programs of problem size $t = 2.5$ and performed on four widely distributed processors. For this N, the precision of the load distribution is great enough to lead to significantly high gains in running times that completely compensate the extra overhead.

Third, we pointed out that a distributed program run with a CCR of 0.01 gains fundamentally by DLB. This speedup grows when the number of processors increases and when the nodes are more widely, or even globally distributed. The last aspect increases the potential speedup because the job runtimes on the different deployed nodes correlate less. Moreover, we illustrate that the gain in running times decreases, as expected, when the CCR increases. Furthermore, it has been demonstrated that in some cases there exists an optimal number of processors.

Fourth, we have investigated the impact of implementing DLB schemes on the running times of SOR in a grid environment. Extensive experimentation in the testbed environment PlanetLab have led to the following conclusions. (1) A significant speedup factor of on average 1.8 can be consistently achieved by implementing DLB instead of the default ELB scheme for a parallel program of size $t = 2.5$ and which runs on 4 processors. This results corresponds with the trace-driven simulation results. (2) Using DLB based on predictions of the job runtimes provides an effective means to react to changes in the performance of the resources used by a parallel application. (3) The relation between the running times of DLB and of ELB and the problem size are both approximately linear. Consequently, this observation indicates that when the CCR gets lower than 0.01, the speedups grow, but is upper bounded. For example, in our experimental setting the upper bound of the speedup equals 1.87.

5

PERFORMANCE EVALUATION OF JOB REPLICATION STRATEGIES

5.1 Introduction

In Chapter 4, the concept of Dynamic Load Balancing (DLB) has been presented. Moreover, research has been done to get insight in the possibilities of DLB. As an alternative of DLB we introduced in Chapter 1 the concept of job replication (JR) in SPMD programs. A complete SPMD program run consists of a number of iterations and within each iteration a number of jobs execute on the processors in parallel. In a JR run all the jobs are replicated a number of times and all samples are distributed over different processors. As soon as the first sample of a specific original job is finished all the other samples are killed. An advantage of JR is that the program is more robust against sudden peaks in job runtimes. A disadvantage is that more jobs have to be executed, which increases the total amount of work. In order to get knowledge about the possibilities of JR and know for which grid environment properties it outperforms DLB, it is essential to get insight in the characteristics of the speedups, and the sensitivity of the speedups if some factors change.

To this end, in this chapter* we first define a model in Section 5.2 to compute the expected iteration times of SPMD programs that apply job replication (JR) on a set of homogeneous processors. The following set of factors are used as input parameters in the model: the number of processors, the number of replications, the average job runtime, the standard deviation of the job runtimes, and a set of three different functions or values that represent the sensitivity of different statistical properties if one of the parameters are changed. Further, we show in Section 5.3, a number of approximations that can easily be applied in the model computations. Moreover, evaluations of the approximations show that those approximations are realistic. In addition, we introduce in Section 5.4 six different sets of model assumptions ranging from less realistic and mathematically simple, which provides insight in the general characteristics and sensitivity of the JR speedups, to realistic and mathematically difficult, which provides a very realistic representation of the real JR implementations. Subsequently, in Section

*This chapter is based on papers [24], and [27].

5.5 we compare the results of the models to the speedup results of trace-driven simulations based on real grid testbed measurements. The results show that the model accurately represents JR in real grid environments. Moreover, in Section 5.6 we perform extensive trace-driven simulation experiments for many different settings on heterogeneous nodes. The results provide insight into the impact of decisions to be made in an experimental setting (e.g., the number of processors, the number of replications) and the impact of the computing environment properties (e.g., the fluctuations in the job runtimes) on the JR speedup. Finally, we present the conclusions.

5.2 The model

In this section, we model SPMD program runs based on JR on a set of homogeneous processors which are processor shared. The processors are homogeneous in the sense that they have the same hardware properties (e.g., same processor speed, hard-disk capacity). The processors are shared, which means that many users use the processors at the same time and they all get an equal part of the processors capacity. We present two computations based on this model that effectively compute the iteration time of different JR runs. The individual job runtimes on the P processors are used as input factors. Moreover, approximations of the expected minimum, maximum, standard deviation of the minimum, standard deviation of the maximum, and the sum of stochastic variables, can easily be added to one of the computations. First, we explain the necessary definitions of the model. Second, two representative examples will be explained. Finally, in the present section the two sets of computations will be discussed.

Let R be the number of samples that exist of each job, P the number of processors available for the parallel program, and $M = \frac{P}{R}$ the number of different original jobs that run at the same time. In this chapter, we mostly use input parameters R and P that are powers of two. Denote r and p as the 2-logs of R and P, and therefore $R = 2^r$ and $P = 2^p$. By definition, within one iteration exactly P original jobs are executed, which implies that the total number of jobs that is executed on all processors within each iteration is $R \times P$. We assume that M is an integer, and we call a group of computers that always receive the same jobs during the execution of all the jobs a *computer group*. As a consequence, the number of computer groups equals M. Hence, each processor in a computer group executes during one iteration in total R different jobs. Processors $1, \ldots, R$ form computer group 1, processors $R+1, \ldots, 2R$ computer group 2, ..., and processors $(\frac{P}{R}-1)R+1, \ldots, P$ constitute computer group $\frac{P}{R}$. The number of synchronization moments in a JR run is the same as for a non-JR run. Further, in order to build a model of a JR iteration, we define

Definition 5.2.1: X := job-runtime distribution with expectation $\mathbb{E}X$, and standard deviation $\sigma(X)$.

The $\mathbb{E}X$, and the $\sigma(X)$ depend on the problem size t, which is defined in Section 2.5. Corresponding to Assumption 2.5.1, in the model the expectation of the job runtimes grows linearly with the job size. In addition, in Section 3.8 we concluded that there exist a linear relation between the expectation and the standard deviation of the job runtimes. Therefore, we conclude that the $\sigma(X)$ indirectly linearly depends on the problem size. Furthermore, we define

Definition 5.2.2: X_{mno} := the processing time of the oth replicate ($o = 1, 2, \ldots, R$) of the nth job ($n = 1, 2, \ldots, R$) on the mth computer group ($m = 1, 2, \ldots, M$), with $X_{mno} \sim X$.

Definition 5.2.3: $V_{r,p}(k)$:= the kth, $k \in 0, \ldots, 2^{rp} - 1$, iteration time of a program run on $P = 2^p$, $p = 0, 1, \ldots$ processors where each job is replicated $R = 2^r$, $r = 0, 1, \ldots, p$ times.

Note that by definition the following holds:

$$R_{it}(t, 2^r, 2^p) := \mathbb{E}(V_{r,p}(k)). \tag{5.1}$$

The relation between the X_{mno} and the $V_{0,0}(k)$ is defined by the following two rules:

$$1)\ X_{mno} := V_{0,0}(k),\ \text{for } k = 0, \ldots, 2^{rp} - 1, \tag{5.2}$$

with $m = \lfloor \frac{k}{2^r} \rfloor mod(2^{p-r}) + 1,\ n = \lfloor \frac{k}{2^p} \rfloor + 1,$ and $o = k\ mod(2^r) + 1$, and

$$2)\ V_{0,0}(k) := X_{mno},\ \text{for } m = 1, \ldots, M,\ n = 1, \ldots, 2^r,\ o = 1, \ldots, 2^r. \tag{5.3}$$

with $k = 2^r(m-1) + 2^p(n-1) + o - 1$.

For simplicity, we do not consider the send and synchronization times in the models presented in this chapter because it is important to focus completely on the job runtimes. The send and synchronization times can be included in the model relatively easy. In Section 5.6, those two types of times are included in the experiments.

Given the above definitions, Figure 5.1 illustrates an example of a run based on job replication with two samples of each job (a '2-JR' run) which is a simplification of Figure 1.3 as provided in Chapter 1, because the send and synchronization times are assumed to be negligible. Below, we get into more detail about the illustrated setting.

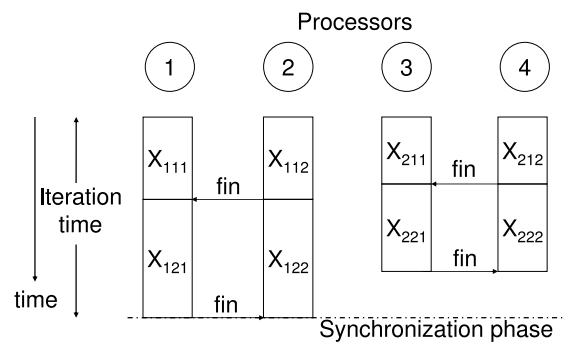


Figure 5.1: Illustration of two times Job Replication on four nodes

In addition, Figure 5.2 represents a 2-JR run with the assumption that all job runtimes of one single processor in one iteration are the same. For example, in this figure the first job on processor 1, X_{111}, and the second job, X_{121}, have the same duration.

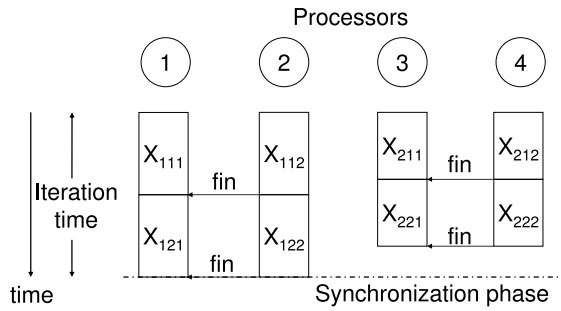


Figure 5.2: Illustration of two times Job Replication with extra assumption

As described above, Figures 5.1 and 5.2 both show the situation for an iteration of a 2-replication run on 4 processors for slightly different sets of assumptions. In addition, the figures illustrate the variable names for these JR run examples. The figures can be interpreted as follows. Each original job and its replications are distributed to 2 (R) processors and during one iteration each processor receives 2 (R) jobs. The first processor that finishes the job sends a message to the other processor(s) which received a copy (copies) of the same job to finish the execution of the current job and start the next job in the sequence. Therefore, the time that the execution of a job and its replications take equals the minimum of their job runtimes. The complete execution time of one processor in one iteration equals the sum of the minimum job runtimes of its computer group. Finally, the total iteration time corresponds to the maximum of all individual processor times. In both figures X_{111} and X_{112} are replicates of the same job. The same holds for the sets X_{211} and X_{212}; X_{121} and X_{122}; and X_{221} and X_{222}. For Figure 5.2 the following equations hold: $X_{111} = X_{121}$, $X_{112} = X_{122}$, $X_{211} = X_{221}$, and $X_{212} = X_{222}$. Further in this section, we extend the two types of models presented above for more processors and higher numbers of replications.

Given Definition 5.2.2 and the above considerations, we are able to derive the following straightforward equation of the iteration time for each possible R and P combination for $t > 0$, $R = 2^r$, $r = 0, 1, \ldots, p$, $P = 2^p$, $p = 1, 2, \ldots$:

$$R_{it}(t, R, P)) = \mathbb{E} \max_{m=1,\ldots,M} \sum_{n=1}^{R} \min_{o=1,\ldots,R} X_{mno}, \tag{5.4}$$

with

$$X_{mno} \sim X, \text{ for } m = 1, \ldots, M,\ n = 1, \ldots, 2^r,\ o = 1, \ldots, 2^r.$$

This implies in combination with (5.3) that for $k = 0, \ldots, 2^{rp-1}$:

$$V_{0,0}(k) \sim X. \tag{5.5}$$

An advantage of equation (5.4) is that it is a compact mathematical description of the computation of an iteration time. A disadvantage is that it is hard to use (5.4) to make approximations of $R_{it}(t, R, P))$. The computation for the iteration time for each R, P-combination as shown in (5.4), can be rewritten in recursive equations in terms of $V_{i,j}(k)$ (see Definition 5.2.3) in the following way for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, 1, \ldots, p$, $p = 0, 1, \ldots$, and $k = 0, \ldots, 2^{rp-1}$:

$$V_{i,j}(k) = \begin{cases} \max\{V_{i,j-1}(2k), V_{i,j-1}(2k+1)\}, & \text{if } i < j, \\ \min\{V_{i-1,j-1}(2k), V_{i-1,j-1}(2k+1)\}, & \text{if } i = j, i \neq r, \\ \sum_{m=0}^{2^r-1} \min \left\{ \begin{array}{l} V_{i-1,j-1}(k2^{r+1}+2m) \\ V_{i-1,j-1}(k2^{r+1}+2m+1) \end{array} \right\}, & \text{if } i = j = r, \\ 0, & \text{if } i > j. \end{cases} \tag{5.6}$$

A disadvantage of (5.6) is that this mathematical description of the iteration-time computation is more complex and harder to understand. This equation has the advantage that it is easy to incorporate approximations in this equation, which is described in the next section. Because of this advantage, we often use this equation in the remainder of this chapter. Note that we officially need to add the r-parameter in the $V_{i,j}(k)$, because it slightly depends on r (the values for $V_{i,j}(k)$ differ for the cases that $i = j = r$ and $i = j \neq r$). However, in order to make the notation not too complex we omit the r parameter.

5.3 Model approximations

In this section, we first introduce approximations which provide a strong basis for the investigations to the speedup characteristics of the next section. Furthermore, we motivate them and investigate the reasonableness of those approximations. Finally, we derive two other approximations from the already introduced ones.

5.3.1 Approximations

Below, we provide an overview of all the assumptions and approximations that have been made. We introduce and motivate the following Assumptions 5.3.1, 5.3.2, and Approximations 5.3.3-5.3.5. The first assumption that has been made is

Assumption 5.3.1: $V_{i,j}(k), V_{i,j}(l)$ are identically distributed, for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, \ldots, p$, $p = 0, 1, \ldots$, $i \leq j$ and $k, l \in 0, \ldots, 2^{rp} - 1$,

The following assumptions 5.3.2–5.3.4 have been made for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, \ldots, p$, $p = 0, 1, \ldots$, $i \leq j$ and $k, l \in 0, \ldots, 2^{rp} - 1$, where $\lfloor \frac{k}{2^r} \rfloor mod(2^{p-r}) \neq \lfloor \frac{l}{2^r} \rfloor mod(2^{p-r})$, or $kmod(2^r) + 1 \neq l \; mod(2^r)$. Those two inequalities correspond according to (5.3) to the situation where the values for m or o of two different X_{mno}'s are different which means that the job runtimes are measured on two different processors.

Assumption 5.3.2: $V_{i,j}(k), V_{i,j}(l)$ are i.i.d.

Approximation 5.3.3: $\mathbb{E} \max\{V_{i,j}(k), V_{i,j}(l)\} \approx \mathbb{E}V_{i,j}(k) + \alpha\sigma(V_{i,j}(k))$, with $0 < \alpha < 1$ and given that $\sigma(V_{i,j}(k)) \leq \mathbb{E}(V_{i,j}(k))$.

Approximation 5.3.4: $\sigma(\max\{V_{i,j}(k), V_{i,j}(l)\}) \approx \beta(i, j+1)\sigma(V_{i,j}(k))$, for $\beta(i, j+1) \geq 0$.

Finally, we approximate the standard deviation of a sum of $V_{i,j}(k)$s, for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, \ldots, p$, $p = 0, 1, \ldots$, and $i \leq j$ by the approximation below. This approximation holds for $V_{i,j}(k)$s measured on the same or on different nodes.

$$\sigma\left(\sum_{k=0}^{2^r-1} V_{i,j}(k)\right) \approx 2^{r\gamma}\sigma(V_{i,j}(0)), \text{ with } \frac{1}{2} < \gamma < 1. \tag{5.7}$$

The dependency among the $V_{i,j}(k)$s is taken into account in the above assumption by the γ: the higher the γ the more dependency. Because of Assumption 5.3.2, $\gamma = \frac{1}{2}$ if those are measured on different nodes. In our setting we observe dependency among the job runtimes because they are measured on the same processor. We derive the following for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, \ldots, p$, $p = 0, 1, \ldots$, $i \leq j$, $k = 0, \ldots, 2^{rp} - 1$, and $\frac{1}{2} < \gamma < 1$ if (5.7) is translated to the setting of this chapter:

Approximation 5.3.5: $\sigma\left(\sum_{l=0}^{2^r-1} V_{i,j}(k + l * 2^p)\right) \approx R^{\gamma}\sigma(V_{i,j}(k)) = 2^{r\gamma}\sigma(V_{i,j}(k))$.

Motivation:
First, Assumption 5.3.1 indicates that all measured job runtimes are identically distributed. The fact that all processors are homogeneous and approximately receive the same load indicates that this assumption is reasonable. Next, we assume in Assumption 5.3.2 that the running times of the jobs in the same iteration step on different processors are independent and identically distributed. In our setting all processors are homogeneous and the expectations of the amount of load on the different processors are the same, which illustrates that this assumption is reasonable. In addition, Approximation 5.3.3 uses the property that the maximum of two i.i.d. variables strongly depends on the expectation and the standard deviation of those variables. The value of α value strongly depends on the shape of the job-runtimes distribution. For the case that the variables are exponentially distributed, the α value equals exactly 0.5. The α-value *theoretically* depends on the r, p, and γ values. However, investigations further in this chapter show that α loosely depends on these values. The results of a few experiments show that this approximation looses its accuracy if the $\sigma(V_{i,j}(k)) > \mathbb{E}(V_{i,j}(k))$ and therefore use this

approximation only if $\sigma(V_{i,j}(k)) \leq \mathbb{E}(V_{i,j}(k))$. Moreover, Approximation 5.3.4 implies that the standard deviation of the maximum of two i.i.d. variables approximately equals the standard deviation of one of those variables multiplied by a factor $\beta(i,j)$. For example, for the case that the variables are exponentially distributed, the $\beta(i,j)$ value equals $\frac{1}{2}\sqrt{5}$ independent of the i, j, r and p. From combining (5.6) and Approximation 5.3.4 follows that $\beta(i,j) = 0$ for $i \geq j$, and moreover that $\beta(i,j) = \beta(r,p)$ if $i = r$ and $j = p$. Investigations further in this section show that the $\beta(r,p)$ values depend on the r, and p for the data collected in Chapter 2. Approximation 5.3.5 shows that the standard deviation of the sum of a sequence of job runtimes on the same processor equals the standard deviation of one individually measured job runtime multiplied with the number of job runtimes in the sum to the power a γ value. In our model we use this approximation to approximate the standard deviation of a sum of job runtimes that are measured on the same processor. The height of the γ value strongly depends on the correlation coefficient between successive job runtimes on the same processor. Theoretically, the γ value ranges from $-\infty$ (complete negative correlation), via $\frac{1}{2}$ (no correlation) to ∞ (complete positive correlation). In practice, $\frac{1}{2}$ is the lower bound, and 1 is the upper bound of the γ value. In general, the γ value depends on the R. Investigations into the γ values for different sums of job-runtime measurements show that these value do not show significant correlation with the R. Therefore, we assume that the γ is constant.

5.3.2 Evaluation of approximations

In the previous section, we have defined several approximations. In this section, we investigate how realistic those approximations are. We know that Approximations 5.3.3–5.3.5 are exact for identical independent exponentially distributed measurements and that the parameters for this case are: $\alpha = \frac{1}{2}$, $\beta(i,j) = \frac{1}{2}\sqrt{5}$ for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, 1, \ldots, p$, $p = 0, 1, \ldots$, $i \leq j$, and $\gamma = \frac{1}{2}$ independent of the i, j, r and p. We performed experiments with the normal distribution and the results show that the approximations are close to exact. Furthermore, we tested the approximations with the 130 real Planetlab datasets which are described in Chapter 2. The analyses are described below.

In the final section of this chapter, we will compare the model results with trace-driven simulations of JR-runs that consist of $I = 2000$ iterations. For that reason we analyze the properties of the averages of 2000 α, $\beta(r,p)$, and γ values. In total we derive 1000 average α values which provides a strong basis for the evaluation. To this end, in order to derive one average value, we randomly select 1000 times two datasets from the 130 datasets. With each selection of two datasets we derive 2000 single α values according to Approximation 5.3.3 and compute their average. Figure 5.3 depicts 1000 average α values. Moreover, we performed the same type of experiments in order to derive a set of 1000 average $\beta(0,1)$ values: we derive the $\beta(0,1)$ values according to Approximation 5.3.4 by dividing the standard deviation of the maximum of two jobs by the standard deviation of one of the jobs. The results are shown in Figure 5.4.

Figure 5.3 illustrates that the α value of a run equals on average 0.51 with a standard deviation of 0.05. We observe that the α values all fluctuate around that value and therefore this result endorses the assumption of one fixed α parameter, which is assumed in Approximation 5.3.3. However, more research needs to be done in order to validate this approximation, which is described below.

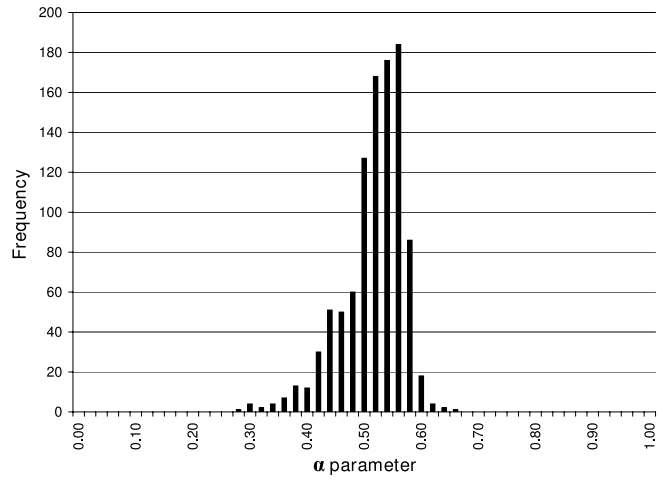


Figure 5.3: Histogram of measured α values

Figure 5.4 illustrates that the $\beta(0,1)$ value of a run equals on average 1.05 with a standard deviation of 0.10. Again these results endorse a fixed $\beta(0,1)$ parameter for the JR situation with $p = 1$ and $r = 0$, as is assumed in Approximation 5.3.4.

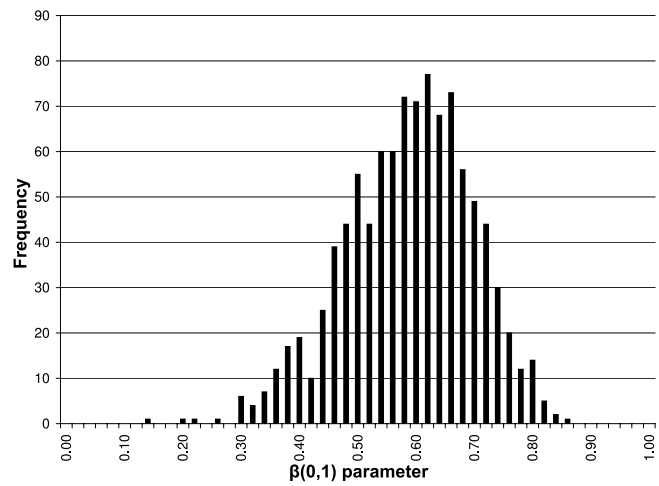


Figure 5.4: Histogram of measured $\beta(0,1)$ values

We repeat the above analyses for each $r = 0, 1, \ldots, 6$ and $p = 1, \ldots, 7$ combination that exists in the model for the parameters α, $\beta(r, p)$, and for γ for each $r = 0, \ldots, 6$. In order to derive the α and $\beta(r, p)$ values, we assume that $\gamma = 1.0$. Tables 5.1, 5.2, and 5.3 depict the fitted values of those parameters for α, $\beta(r, p)$, and γ respectively. Each value in the tables represents an average of 1000 averages that are all derived from two pairs of datasets consisting of 2000 iterations. Additionally, we derived the α and $\beta(r, p)$ values given that $\gamma = 0.81$. We do not represent the results of the tables of α, and $\beta(r, p)$ with $\gamma = 0.81$ because these were not significantly different from the above tables.

$r \backslash p$	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0.51	0.51	0.49	0.45	0.42	0.38	0.35
1		0.54	0.55	0.54	0.55	0.55	0.54
2			0.53	0.54	0.53	0.52	0.53
3				0.52	0.50	0.49	0.49
4					0.50	0.47	0.42
5						0.46	0.47
6							0.43

Table 5.1: Values of α for $r = 0, \ldots, 6$, $p = 1, \ldots, 7$, and $\gamma = 1.0$

$r \backslash p$	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	1.05	1.07	1.10	1.14	1.22	1.26	1.26
1		0.95	1.01	0.99	0.97	0.96	0.96
2			0.80	0.94	0.99	1.00	1.00
3				0.71	0.82	0.89	0.95
4					0.67	0.83	0.84
5						0.62	0.86
6							0.53

Table 5.2: Values of $\beta(r, p)$ for $r = 0, \ldots, 6$, $p = 1, \ldots, 7$, and $\gamma = 1.0$

r	**1**	**2**	**3**	**4**	**5**	**6**
γ	0.20	0.17	0.18	0.15	0.18	0.21

Table 5.3: Values of γ for $r = 1, \ldots, 6$

We observe from Table 5.1 that the difference between the α values for different r and p values is minor and it seems that the r and the p do not have a huge impact on the α parameter. As can be seen in Table 5.3, the same holds for the γ values. On the contrary, the difference between the β values for different r and p values is noticeable. The above observations show that it is necessary to investigate the potential

relations in more detail. To this end, we perform investigations to those relations in the next three sections. We first investigate the impact of r and p on the α, $\beta(r,p)$, and γ. Subsequently, we analyze the impact of γ on the α, and $\beta(r,p)$. Finally, we fit α, $\beta(r,p)$, and γ into functions or fixed values.

Impact of r and p on the α, $\beta(r,p)$, and γ

In this section, we first investigate the correlation between the r and p and the α, β, and γ. Second, we perform analyzes on the impact of the significant correlation coefficients. To perform detailed analyses on the question whether the different parameters can be estimated by a fixed value or by a function, we perform investigations to the correlation-coefficients, as defined in Section 2.5. Table 5.4 shows the expected correlation coefficients between the parameters α, $\beta(r,p)$, γ, and r, p which quantifies their linear relations. For example, to compute the expected correlation coefficient between the α parameter and the r, we first fix the value of p, derive the correlation coefficient, and repeat these steps for all possible p values. Finally, we compute the weighted average of those correlation coefficients where the weights equal the number of possible r values for this p.

Correlation	$\boldsymbol{r}$	$\boldsymbol{p}$
$\boldsymbol{\alpha}$	-0.31	-0.70
$\boldsymbol{\beta(r,p)}$	-0.94	0.65
$\boldsymbol{\gamma}$	0.13	

Table 5.4: Correlation between α, $\beta(r,p)$, γ and r, p

The results of the parameters correlation investigations illustrate that there is no significant dependency between α and r, and between γ and r. However, the following pairs of variables are negatively correlated: the α and the p, and the $\beta(r,p)$ and the r. The $\beta(r,p)$ and the p are positively correlated. These results show that the parameters that have high correlation with r and p can be described as linear functions of those parameters. The above result for the $\beta(r,p)$ values corresponds with earlier observations, however, we did not expect a high correlation coefficient between the α and the p. This heightens the need for further research on the actual impact of the p on the α. Therefore, we compute the derivative of the α to the p. To compute this, we first compute the average value of α for each value of p. Next, we derive the least-squares trendline of the α against the averaged ps. Finally, the derivative of the α to the p is the derivative of the trendline. For completeness, in addition we compute the derivative to r. Both derivatives are shown in Table 5.5.

Derivative	Value
$\frac{\delta(\alpha)}{\delta(r)}$	-0.008
$\frac{\delta(\alpha)}{\delta(p)}$	-0.009

Table 5.5: Derivative of α against r, and p

We conclude from Table 5.5 and the above investigations to the α characteristics that p indeed shows a significant correlation with α, but that the impact of this parameter is relatively low (-0.009). Consequently, we do not take the r and p parameters into account in the α approximation equations. The observation that the α and γ parameters almost remain the same if the number of minima, maxima, or sums, which is represented by the i and j in $V_{i,j}(k)$ and the choice of the γ, changes is rather surprising.

Impact of γ on the α, and $\beta(r,p)$

In this section, we first perform t-tests to investigate the impact of the γ value on the α and the $\beta(r,p)$ parameters and second the impact of the relations with significant correlations. We first generated a sequence of α-value pairs which are generated from Approximation 5.3.3. Each two α values that constitute a pair are generated with the same values for $r = 0,\ldots,6$, and $p = 1,\ldots,7$, and with two different γ values: within each pair the first value is generated with $\gamma = 0.81$, and the second with $\gamma = 1.00$. The null hypothesis is defined as follows: the α values with corresponding r and p values are independent on the γ values. The t-test uses a double tail distribution and we reject the null hypothesis if the p-value is lower than 0.025 or higher than 0.975. The p-value of the t-test is 0.017, which means that we reject the null hypothesis. This means that the α parameter depends on the γ parameter. We perform the same test for the $\beta(r,p)$ value. The p-value for the $\beta(r,p)$-parameter is 0.80 which means that we do not reject the null hypothesis. Therefore, we assume that the γ does not have a significant impact on the $\beta(r,p)$. The above observations heighten the need for further investigations on the actual impact of the relation between the γ value and the α parameter.

In order to investigate the possible relation between the γ and the α we compute the fraction between the α values generated with $\gamma = 0.81$ and $\gamma = 1.00$. The 95%-reliability interval of this proportion is $[-1.29, 0.78]$ with an expectation of -0.25. We conclude that the γ indeed has a influence on the α characteristics, but that we are not able to catch this impact in a simple formula due to the significant fluctuations of the impact which is indicated by the reliability interval. For this reason, we further assume that α is independent of the γ.

Fits of α, $\beta(r,p)$, and γ

As shown above, the α and the γ can be fitted by fixed values. Because of the high correlation coefficients between the $\beta(r,p)$ and the r and the p, we expect that the function $(a_1 + b_1 r)(a_2 + b_2 p)$, where a_1, b_1, a_2, and $b_2 \in \mathbb{R}$ have to be fitted, delivers a reliable fit of $\beta(r,p)$. Table 5.6 describes the best function or fixed value fits of the

three parameters, based on all the above investigations and observations. Moreover, the mean absolute error (MAE) and the square root of the mean squared error (RMSE) between the value and the parameter values derived from the data are shown.

Parameter	Fitted value or function	Mean absolute error	RMSE
α	0.50	0.06	0.06
$\beta(i,j), i < j$	(-0.14 i + 1.54)(0.02 j + 0.63)	0.04	0.07
γ	0.81	0.02	0.02

Table 5.6: Properties of α, $\beta(r,p)$, and γ

The low values for the MAE and the RMSE of the fitted values or functions in Table 5.6 indicate the high accuracy of the fits. Therefore, we conclude that the approximations of the parameters by the values or functions which are provided by the table are valid.

5.3.3 Implications of the assumptions and approximations

We introduced approximations for the expectation and the standard deviation of the maximum of two independent job runtimes. In this section, we derive formulas for approximations of the expectation and the standard deviation of the *minimum* of two independent job runtimes, given these previously introduced assumptions and approximations. First, we derive an approximation for the expectation of the minimum.

Approximation 5.3.6: If $V_{i,j}(k)$, and $V_{i,j}(l)$ are i.i.d., $\mathbb{E}\max\{V_{i,j}(k), V_{i,j}(l)\} \approx \mathbb{E}V_{i,j}(k) + \alpha\sigma(V_{i,j}(k))$, and $\sigma(V_{i,j}(k)) \leq \mathbb{E}(V_{i,j}(k))$ for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, 1, \ldots, p$, $p = 0, 1, \ldots$, $i \leq j$, $k, l = 0, \ldots, 2^{rp} - 1$, and $0 < \alpha < 1$, then,

$$\mathbb{E}\min\{V_{i,j}(k), V_{i,j}(l)\} \approx \mathbb{E}V_{i,j}(k) - \alpha\sigma(V_{i,j}(k)).$$

Motivation:
The following equations hold for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, 1, \ldots, p$, $p = 0, 1, \ldots$, $i \leq j$, $k, l = 0, \ldots, 2^{rp} - 1$, and $0 < \alpha < 1$:

$$\begin{aligned}
\mathbb{E}\min\{V_{i,j}(k), V_{i,j}(l)\} &= \mathbb{E}\min\{V_{i,j}(k), V_{i,j}(l)\} + \mathbb{E}\max\{V_{i,j}(k), V_{i,j}(l)\} \\
&\quad - \mathbb{E}\max\{V_{i,j}(k), V_{i,j}(l)\} \\
&= \mathbb{E}V_{i,j}(k) + \mathbb{E}V_{i,j}(l) - \mathbb{E}\max\{V_{i,j}(k), V_{i,j}(l)\} \\
&\approx \mathbb{E}V_{i,j}(k) + \mathbb{E}V_{i,j}(l) - (\mathbb{E}V_{i,j}(k) + \alpha\sigma(V_{i,j}(k))) \\
&= \mathbb{E}V_{i,j}(k) - \alpha\sigma(V_{i,j}(k)). \qquad (5.8)
\end{aligned}$$

The first step is trivial, the second uses the property that the sum of two variables trivially equals the sum of the minimum and the maximum of the those variables, the third uses Assumption 5.3.3, and the last step erases the variables that cancel each other out. □

Second, we derive an approximation of the standard deviation of the minimum of two variables.

Approximation 5.3.7: If $V_{i,j}(k)$, and $V_{i,j}(l)$ are i.i.d., $\mathbb{E}\max\{V_{i,j}(k), V_{i,j}(l)\} \approx \mathbb{E}V_{i,j}(k) + \alpha\sigma(V_{i,j}(k))$, and $\sigma(\max\{V_{i,j}(k), V_{i,j}(l)\}) \approx \beta(i, j+1)\sigma V_{i,j}(k)$, for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, 1, \ldots, p$, $p = 0, 1, \ldots, i \leq j$, $k, l = 0, \ldots, 2^{rp} - 1$, $0 < \alpha < 1$, and $\beta(i, j+1) > 0$, then,

$$\sigma^2(\min\{V_{i,j}(k), V_{i,j}(l)\}) = (2 - 2\alpha^2 - \beta(i, j+1)^2)\sigma^2(V_{i,j}(k)).$$

Motivation:
All the equations below hold for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 0, 1, \ldots, p$, $p = 0, 1, \ldots, i \leq j$, $k, l = 0, \ldots, 2^{rp} - 1$, $0 < \alpha < 1$, and $\beta(i, j+1) > 0$. The correlation coefficient, $r_{\min,\max}$, between the minimum and the maximum of the two variables $V_{i,j}(k)$, and $V_{i,j}(l)$ can be derived by substituting $\min\{V_{i,j}(k), V_{i,j}(l)\}$ (abbreviated by $\min$), and $\max\{V_{i,j}(k), V_{i,j}(l)\}$ (abbreviated by $\max$) in the general definition of the correlation coefficient, Definition 2.5.10:

$$r_{\min,\max} := \frac{\mathbb{E}(\min\max) - \mathbb{E}(\min)\mathbb{E}(\max)}{\sigma(\min)\sigma(\max)}. \tag{5.9}$$

Consequently, by substituting

$$\mathbb{E}(\min\{V_{i,j}(k), V_{i,j}(l)\}\max\{V_{i,j}(k), V_{i,j}(l)\}) = \mathbb{E}(V_{i,j}(k)V_{i,j}(l)) = (\mathbb{E}(V_{i,j}(k)))^2 \tag{5.10}$$

in (5.9), we derive

$$r_{\min,\max} := \frac{(\mathbb{E}V_{i,j}(k))^2 - \mathbb{E}\min\{V_{i,j}(k), V_{i,j}(l)\}\mathbb{E}\max\{V_{i,j}(k), V_{i,j}(l)\}}{\sigma(\min\{V_{i,j}(k), V_{i,j}(l)\})\sigma(\max\{V_{i,j}(k), V_{i,j}(l)\})}. \tag{5.11}$$

Furthermore, we derive the following for $2\sigma^2(V_{i,j}(k))$:

$$\begin{aligned} 2\sigma^2(V_{i,j}(k)) &= \sigma^2(V_{i,j}(k) + V_{i,j}(l)) \\ &= \sigma^2(\min\{V_{i,j}(k), V_{i,j}(l)\} + \max\{V_{i,j}(k), V_{i,j}(l)\}). \end{aligned} \tag{5.12}$$

Moreover, by substituting the following variables $\min\{V_{i,j}(k), V_{i,j}(l)\}$ and $\max\{V_{i,j}(k), V_{i,j}(l)\}$ in the equation of the variance of the sum of two independent stochastic variables, Property 2.5.11, we derive:

$$\begin{aligned} &\sigma^2(\min\{V_{i,j}(k), V_{i,j}(l)\} + \max\{V_{i,j}(k), V_{i,j}(l)\}) \\ &= \sigma^2(\min\{V_{i,j}(k), V_{i,j}(l)\}) + \sigma^2(\max\{V_{i,j}(k), V_{i,j}(l)\}) \\ &\quad + 2r_{\min,\max}\sigma(\min\{V_{i,j}(k), V_{i,j}(l)\})\sigma(\max\{V_{i,j}(k), V_{i,j}(l)\}). \end{aligned} \tag{5.13}$$

Equations (5.12) and (5.13) can be rewritten in the following equation:

$$\begin{aligned} &\sigma^2(\min\{V_{i,j}(k), V_{i,j}(l)\}) \\ &= 2\sigma^2(V_{i,j}(k)) - \sigma^2(\max\{V_{i,j}(k), V_{i,j}(l)\}) \\ &\quad - 2r_{\min,\max}\sigma(\min\{V_{i,j}(k), V_{i,j}(l)\})\sigma(\max\{V_{i,j}(k), V_{i,j}(l)\}). \end{aligned} \tag{5.14}$$

Consequently, we derive the following equation for $\sigma^2(\min\{V_{i,j}(k), V_{i,j}(l)\})$ by substituting (5.11), (5.3.4), and (5.3.3) in (5.14) and solving the equation:

$$\sigma^2(\min\{V_{i,j}(k), V_{i,j}(l)\}) = 2(1 - \alpha^2 - \beta(i, j+1)^2)\sigma^2(V_{i,j}(k)). \tag{5.15}$$

Which corresponds to:

$$\sigma(\min\{V_{i,j}(k), V_{i,j}(l)\}) = \sqrt{2 - 2\alpha^2 - \beta(i, j+1)^2}\sigma(V_{i,j}(k)). \tag{5.16}$$

□

Note that in our model $\beta(i, j)$ with $i < j$ represent the $\beta(i, j)$ parameters that belong to the equation that approximates the standard deviation of the maximum. From combining (5.6) and Approximation 5.3.4 follows that $\beta(i, j) = 0$ for $i \geq j$. Furthermore, for the equation that computes the minimum of the standard deviation, we only need the $\beta(i, j)$ with $i = j$ which can be derived from the $\beta(i-1, j)$ values by Approximation 5.3.7. Therefore, to prevent that the model has too many parameters we define

Definition 5.3.8: $\beta(i, i) := \sqrt{2 - 2\alpha^2 - \beta(i-1, i)^2}$, for $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $p = 0, 1, \ldots$, $0 < \alpha < 1$, and $0 \leq \beta(i-1, i) \leq 2 - 2\alpha^2$.

Hence, we derive from (5.16) and Definition 5.3.8 the following equation for the standard deviation of the minimum which is used in the remainder of the present chapter, for $i = 0, \ldots, r,$ and $0 \leq \beta(i-1, i) \leq 2 - 2\alpha^2$:

$$\sigma(\min\{V_{i,i}(k), V_{i,i}(l)\}) = \beta(i+1, i+1)\sigma(V_{i,i}(k)). \tag{5.17}$$

5.4 Theoretical analysis of speedups

In this section, we derive formulas for the speedups of replicating on homogeneous nodes by analyzing six different theoretical models, ranging from less realistic and mathematically simple to realistic and mathematically difficult. Table 5.7 provides an overview of the choices of the different parameters in the different models. Denote $a_1, b_1, a_2, b_2 \in \mathbb{R}$.

Model	**Values**			
	α	$\beta(i,j),\ \forall i,j : i < j$	$\beta(i,j),\ \forall i,j : i = j$	γ
1.1	$\frac{1}{2}$	1	$\frac{1}{2}$	1
1.2	$\in (0,1)$	$\beta_1, \beta_1 \in [0, \sqrt{2-2\alpha^2}]$	$\beta_2,\ \beta_2 = \sqrt{2 - 2\alpha^2 - {\beta_1}^2}$	1
1.3a	$\in (0,1)$	$(a_1 + b_1 i)(a_2 + b_2 j)$	$\sqrt{2 - 2\alpha^2 - \beta(i-1,j)^2}$	1
1.3b	$\in (0,1)$	$\in [0, \sqrt{2-2\alpha^2}]$	$\sqrt{2 - 2\alpha^2 - \beta(i-1,j)^2}$	1
2.1	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
2.2	$\in (0,1)$	$\beta_1, \beta_1 \in [0, \sqrt{2-2\alpha^2}]$	$\beta_2,\ \beta_2 = \sqrt{2 - 2\alpha^2 - {\beta_1}^2}$	$\in [\frac{1}{2}, 1]$
2.3a	$\in (0,1)$	$(a_1 + b_1 i)(a_2 + b_2 j)$	$\sqrt{2 - 2\alpha^2 - \beta(i-1,j)^2}$	$\in [\frac{1}{2}, 1]$
2.3b	$\in (0,1)$	$\in [0, \sqrt{2-2\alpha^2}]$	$\sqrt{2 - 2\alpha^2 - \beta(i-1,j)^2}$	$\in [\frac{1}{2}, 1]$

Table 5.7: Values of α, $\beta(r, p)$, and γ for the six models

As can be seen in this table, models 1.1 and 2.1 assume for the ease of computations that the α equals $\frac{1}{2}$, which corresponds to the result in Table 5.6, the $\beta(i,j)$ for $i < j$ equals 1 which means that the formula that approximates the standard deviation of the maximum equals to $\sigma(\max\{V_{i,j}(k), V_{i,j}(l)\}) \approx \sigma V_{i,j}(k)$, and $\beta(i,j)$ for $i = j$ equals $\frac{1}{2}$ which means that the formula that computes the standard deviation of the minimum equals to $\sigma(\min\{V_{i,j}(k), V_{i,j}(l)\}) \approx \frac{1}{2}\sigma V_{i,j}(k)$. As can be seen in Approximation 5.3.7, this last approximation is not completely correct because $\sqrt{2 - \frac{1}{2} - \frac{1}{2}^2} = \frac{1}{2}\sqrt{5} \neq 1$, but nevertheless the values are still reasonable. Advantages of applying these values are the simplicity in the computations and the insight that can be gained by the simplicity of the resulting formulas. Models 1.1 and 2.1 are closely related to models for JR run-times with exponentially distributed job runtimes. Differences between those types of models are that (1) $\beta(i,j) = 1, i < j$ for models 1.1 and 1.2, and $\beta(i,j) = \frac{1}{2}\sqrt{5}, i < j$ for a model with exponential job runtimes, and (2) the c_X in models 1.1 and 1.2 can take every value above 0, and for exponential distributed stochastics it is known that $c_X = 1$. This close relation is illustrated further in this section. Models 1.2 and 2.2 use a fixed α that can take values between 0 and 1. Because of the reason that the value of α depends on the shape of the underlying job-runtime distribution, relaxation of the constraint that α must equal $\frac{1}{2}$ (in models 1.2 and 2.2 it can adopt to each value between 0 and 1) leads to a more realistic model. In addition, those models assume that all $\beta(i,j)$ equal a fixed value in $[0, \sqrt{2 - 2\alpha^2}]$ which can be adapted to the shape of the job-runtime distribution, instead of the beforehand chosen fixed values 1 and $1/2$ that have been chosen for the ease of computation. Moreover, models 1.3a, 1.3b, 2.3a and 2.3b are even more realistic, because the $\beta(i,j)$ value is considered as a function of i, and j (i.e., r and p). This realistic dependency is an additional feature of these models. Models 1.3a and 2.3a assume that the $\beta(i,j)$s depend on i and j according to the given function, which is already presented in Table 5.6. In models 1.3b and 2.3b every $\beta(i,j)$ can take every value in $[0, \sqrt{2 - 2\alpha^2}]$. Finally, models 1.1-1.3 differ from 2.1-2.3 in the assumption of the γ parameter. In the first set of models, the γ parameter is assumed to be 1, and in the second set as a parameter that can be fixed on every value between $1/2$ and 1 depending on the correlation between successive job runtimes. In Section 5.2 two examples have been illustrated by Figures 5.1 and 5.2 to explain the difference between those sets of models.

In the paragraphs below, we derive with the above described model the speedups for JR runs with different numbers of processors and different numbers of replications, given Assumptions 5.3.1 - 5.3.2, Approximations 5.3.3 - 5.3.5 and the assumed values shown in Table 5.7. For all of the models we analyze the following cases: (1) the iteration time of a 1-JR run, which equals an ELB iteration time, (2) the iteration time of a JR-run with R ($1 < R < P$) replications, (3) the iteration time of a P-JR-run which implies that each processor receives a replication of each job and each iteration consists of P iteration steps, (4) the speedup formula for R-JR runs, (5) the speedups of P-JR runs, (6) if possible to derive, the optimal number of replications, R^*, and (7) if possible to derive, the speedup formula for R^*-JR runs. For models 1.1 and 2.1 we moreover show the derivation of the estimation formula for the iteration times and the speedups for the JR runs on 2 and 4 processors. Those formulas for the other models are derived in the same way.

5.4.1 Model 1.1

In this section, we derive formulas for the iteration times of replicating $R = 1, 2$ times on 2 processors, $R = 1, 2, 4$ times on 4 processors, and $R = 2^r$, $r = 0, 1, \ldots, p$ times on $P = 2^p$, $p = 0, 1, \ldots$ processors on the basis of model 1.1, which is described in Table 5.7. In addition, we obtain formulas for the speedup with input parameters R, P, and c_X. Those parameters together represent the key properties of the experimental setting. Furthermore, we deduce the optimal number of replications as a function of P and c_X, and moreover, the corresponding speedup. We analyze the resulting formulas on the basis of graphical representations. We notice that all equations in this model hold for problem size $t > 0$, which is described in Section 2.5 and for $\sigma(X) \leq \mathbb{E}(X)$ (i.e., $0 \leq c_X \leq 1$) according to the domain of the approximations, defined in Section 5.3.

Replicating on two processors

Given that there are two homogeneous processors in the resource set, we derive the following iteration time if no replication has been applied:

$$\begin{aligned} &R_{it}(t,1,2) = R_{it}(t,2^0,2^1) = \mathbb{E}(V_{0,1}(0)) = \mathbb{E}\max\{V_{0,0}(0), V_{0,0}(1)\} \\ &\approx \mathbb{E}(V_{0,0}(0)) + \frac{1}{2}\sigma(V_{0,0}(0)) = \mathbb{E}(X) + \frac{1}{2}\sigma(X). \end{aligned} \tag{5.18}$$

The first step is trivial, the second is according to (5.1), the third uses (5.6) and the resulting formula equals $\mathbb{E}\max\{X_{111}, X_{211}\}$ according to the X_{mno} notation, the fourth applies Approximation 5.3.3, and the fifth uses (5.5).
For the 2 replicating setting we derive the following expected iteration time:

$$\begin{aligned} &R_{it}(t,2,2) = R_{it}(t,2^1,2^1) = \mathbb{E}(V_{1,1}(0)) \\ &= \mathbb{E}\sum_{m=0,1} \min\{V_{0,0}(2m), V_{0,0}(2m+1)\} \\ &\approx 2\mathbb{E}(V_{0,0}(0)) - 2 \times \frac{1}{2}\sigma(V_{0,0}(0)) \\ &= 2\mathbb{E}(X) - 2 \times \frac{1}{2}\sigma(X) = 2\mathbb{E}(X) - \sigma(X). \end{aligned} \tag{5.19}$$

The first step is trivial, the second is according to (5.1), the third applies (5.6) and the result equals $\mathbb{E}\sum_{m=1,2}\min\{X_{m11}, X_{m12}\})$ in the notation of the X_{mno}, the fourth uses Approximation 5.3.6, the fifth applies (5.5), and the sixth is trivial.
For a resource set with two processors speedup can be obtained by replicating if:

$$\begin{aligned} &\text{Speedup } R(t,2,2) > 1 \Leftrightarrow E(t,2) > R_{it}(t,2,2) \\ &\Leftrightarrow V_{0,1}(0) > V_{1,1}(0) \\ &\Leftrightarrow \mathbb{E}\max\{V_{0,0}(0), V_{0,0}(1)\} > 2\mathbb{E}\min\{V_{0,0}(0), V_{0,0}(1)\} \\ &\sim \frac{\sigma(X)}{\mathbb{E}(X)} = \frac{2}{3} < c_X \leq 1. \end{aligned} \tag{5.20}$$

The first step is according to the definition of the speedup in Section 2.5, the second uses (5.1), the third is according to 5.6, and the fourth makes use of Approximations 5.3.3 and 5.3.6 and Definition 2.5.9 of the coefficient of variation of X, c_X.
Consequently, the optimal number of replications can be estimated as follows:

$$R^* \approx \begin{cases} 1, & \text{if } 0 \leq c_X \leq \frac{2}{3}, \\ 2, & \text{if } \frac{2}{3} < c_X \leq 1. \end{cases} \tag{5.21}$$

According to model 1.1, which is described in Section 5.2, we derive that the speedup of a two-replication run on two processors equals

$$\begin{aligned} &\text{Speedup } R(t,2,2) = \text{Speedup } R(t,2^1,2^1) = \frac{V_{0,1}(0)}{V_{1,1}(0)} \\ &= \frac{\mathbb{E}\max\{V_{0,0}(0), V_{0,0}(1)\}}{2\mathbb{E}\min\{V_{0,0}(0), V_{0,0}(1)\}} \approx \frac{\mathbb{E}(X) + \frac{1}{2}\sigma(X)}{2(\mathbb{E}(X) - \frac{1}{2})} = \frac{2 + c_X}{4 - 2c_X}. \end{aligned} \tag{5.22}$$

The first step is trivial, the second is according to the speedup definition in Section 2.5 and additionally to (5.1), the third applies (5.6) for the numerator and the denominator, the fourth uses Approximations 5.3.3 and 5.3.6 and moreover property (5.5), and the final step makes use of definition (2.5.9) of the correlation coefficient.

Replicating on four processors

In this section we derive approximations of the speedups gained by replicating on four processors. First, we derive the estimation of the iteration time for the non-replication case:

$$\begin{aligned} &R_{it}(t,1,4) = R_{it}(t,2^0,2^2) = \mathbb{E}(V_{0,2}(0)) = \mathbb{E}\max\{V_{0,1}(0), V_{0,1}(1)\} \\ &\approx \mathbb{E}(V_{0,1}(0)) + \frac{1}{2}\sigma(V_{0,1}(0)) \\ &= \mathbb{E}\max\{V_{0,0}(0), V_{0,0}(1)\} + \frac{1}{2}\sigma(\max\{V_{0,0}(0), V_{0,0}(1)\}) \\ &\approx \mathbb{E}(V_{0,0}(0)) + \frac{1}{2}\sigma(V_{0,0}(0)) + \frac{1}{2}\sigma(V_{0,0}(0)) = \mathbb{E}(X) + \sigma(X). \end{aligned} \tag{5.23}$$

The first step is trivial, the second is according to (5.1), the third uses (5.6), the fourth uses Approximation 5.3.3, the fifth applies (5.6) and derived formula equals $\mathbb{E}\max\{X_{111}, X_{211}\} + \frac{1}{2}\sigma(\max\{X_{111}, X_{211}\})$ in the notation of the X_{mno}, the sixth uses Approximation 5.3.3 again, and the last is according to (5.5).
The expected iteration time of a 2-replication run on 4 processors is

$$\begin{aligned} &R_{it}(t,2,4) = R_{it}(t,2^1,2^2) = \mathbb{E}(V_{1,2}(0)) = \mathbb{E}\max\{V_{1,1}(0), V_{1,1}(1)\} \\ &\approx \mathbb{E}(V_{1,1}(0)) + \frac{1}{2}\sigma(V_{1,1}(0)) \\ &\approx 2\mathbb{E}(X) - \sigma(X) + \frac{1}{2}\sigma\left(\sum_{m=0,1} \min\{V_{0,0}(2m), V_{0,0}(2m+1)\}\right) \\ &\approx 2\mathbb{E}(X) - \sigma(X) + \frac{1}{2} \times 2 \times \frac{1}{2}\sigma(V_{0,0}(0)) = 2\mathbb{E}(X) - \frac{1}{2}\sigma(X). \end{aligned} \tag{5.24}$$

The first step is trivial, the second applies (5.1), the third uses (5.6), the fourth uses Approximation 5.3.3, the fifth applies for the left side (5.19), which approximates $\mathbb{E}(V_{i,j}(0))$ for $i = j = r = 1$, and for the right side (5.6), the sixth uses Approximations 5.3.5 and 5.3.7, and the last step is according to (5.5) and applies some trivial computations.
Next, we derive the approximation for the iteration time of a 4-JR run:

$$\begin{aligned}
&R_{it}(t,4,4) = R_{it}(t,2^2,2^2) = \mathbb{E}(V_{2,2}(0)) \\
&= \mathbb{E}\sum_{m=0}^{3} \min\{V_{1,1}(2m), V_{1,1}(2m+1)\} \approx \mathbb{E}4(V_{1,1}(0)) - 4 \times \frac{1}{2}\sigma(V_{1,1}(0)) \\
&= \mathbb{E}4\min\{V_{0,0}(0), V_{0,0}(1)\} - 2\sigma(\min\{V_{0,0}(0), V_{0,0}(1)\}) \\
&\approx 4\mathbb{E}(V_{0,0}(0)) - 4 \times \frac{1}{2}\sigma(V_{0,0}(0)) - 2 \times \frac{1}{2}\sigma(V_{0,0}(0)) \\
&= 4\mathbb{E}(X) - 3\sigma(X). \qquad (5.25)
\end{aligned}$$

The first step is trivial, the second applies (5.1), the third uses (5.6), the fourth uses Approximation 5.3.6, the fifth applies (5.6), the sixth uses Approximations 5.3.6 and 5.3.7, and the last step is according to (5.5) and some trivial computations. Note that we can not use (5.19) in the fifth step, due to the fact that for this equation $i = j = 1 \neq r = 2$.

In the remainder of this section, we do not present the details of the computations. The computations follow mainly the same steps as the computations above. Given the iteration times above, we obtain that replicating 2 times leads to a speedup compared to ELB if:

$$\frac{2}{3} < c_X \leq 1. \qquad (5.26)$$

The speedup of a 2-JR run on 4 processors compared to ELB equals

$$\text{Speedup } R(t,2,4) = \text{Speedup } R(t,2^1,2^2) = \frac{1+c_X}{2-\frac{1}{2}c_X}. \qquad (5.27)$$

Replicating 4 times leads to speedup compared to the 2-replication run if:

$$\frac{4}{5} < c_X \leq 1. \qquad (5.28)$$

The speedup for replicating 4 times on 4 processors is

$$\text{Speedup } R(t,4,4) = \text{Speedup } R(t,2^2,2^2) = \frac{R_{it}(t,2^0,2^2)}{R_{it}(t,2^2,2^2)} = \frac{1+c_X}{4-3c_X}. \qquad (5.29)$$

As a result, we advice the following numbers of replications for given c_X, $0 \leq c_X \leq 1$:

$$R^* \approx \begin{cases} 1, & \text{if } 0 \leq c_X \leq \frac{2}{3}, \\ 2, & \text{if } \frac{2}{3} < c_X \leq \frac{4}{5}, \\ 4, & \text{if } \frac{4}{5} < c_X \leq 1. \end{cases} \qquad (5.30)$$

Replicating $R = 2^r$ times on $P = 2^p$ processors

In this section, we derive formulas for estimations of the iteration times and speedups of replicating $R = 2^r$, $r = 0, 1, \ldots, p$, times on $P = 2^p$, $p = 0, 1, \ldots$, processors and fixed values of parameters α, $\beta(i, j)$ and γ, which are specified in Table 5.7. In order to obtain those formulas for general R and P we extend the computations for the settings with 2 and 4 processors. The descriptions of the computations in this section contain fewer details than the explanations about the computations in the sections above. We use the same assumptions and approximations from Section 5.3 and follow the same type of steps as in the previous computations.

The following approximation for the ELB iteration time has been derived for $P = 2^p$, $p = 0, 1, \ldots$:

$$R_{it}(t, 1, P) = R_{it}(t, 2^0, 2^p) \approx \mathbb{E}(X) + \frac{1}{2}p\sigma(X). \tag{5.31}$$

In the notation of the X_{mno}, $R_{it}(t, 1, P)$ can be rewritten to $\mathbb{E}\max\{X_{111}, X_{211}, \ldots, X_{2^p 11}\}$.
For a JR-run on $P = 2^p, p = 0, 1, \ldots$ processors given $R = 2^r, r = 0, 1, \ldots, p$ replications the expected iteration time equals

$$R_{it}(t, R, P) = R_{it}(t, 2^r, 2^p) \approx 2^r\mathbb{E}(X) + (1 - 2^r + \frac{1}{2}p - \frac{1}{2}r)\sigma(X). \tag{5.32}$$

Consequently, for the special case that we replicate $R = P = 2^p$, $p = 0, 1, \ldots$ times on P processors, the iteration time estimation is for $0 \leq c_X \leq 1$

$$R_{it}(t, P, P) = R_{it}(t, 2^p, 2^p) \approx 2^p\mathbb{E}(X) - (2^p - 1)\sigma(X), \tag{5.33}$$

and for $c_X > 1$ we expect that the iteration time approximately equals

$$R_{it}(t, P, P) = R_{it}(t, 2^p, 2^p) > \approx 0. \tag{5.34}$$

In the X_{mno} notation, the $R_{it}(t, P, P)$ can be rewritten to $2^p\mathbb{E}\min\{X_{111}, X_{112}, \ldots, X_{112^p}\}$.
Formula (5.31) over (5.32) equals the speedup that can be gained by a $R = 2^r$, $r = 0, 1, \ldots, p$,-replication run on $P = 2^p$, $p = 0, 1, \ldots$, processors, which can be rewritten to:

$$\text{Speedup } R(t, R, P) = \text{Speedup } R(t, 2^r, 2^p) \approx \frac{2 + pc_X}{2^{r+1} + c_X\,(2 - 2^{r+1} + p - r)}. \tag{5.35}$$

This formula derives for every c_X, $0 \leq c_X \leq 1$, and the given values for r and p a speedup value between 0 and infinity.
The following speedup of maximal replication ($R = P$) for $P = 2^p$, $p = 0, 1, \ldots$:

$$\text{Speedup } R(t, P, P) = \text{Speedup } R(t, 2^p, 2^p) \approx \frac{1 + \frac{1}{2}pc_X}{(1 - c_X)2^p + c_X}. \tag{5.36}$$

Those formulas provide a tremendous insight in the properties of the speedups and the iteration times for different sets of parameters. Moreover, the optimal number of replications can be derived and the speedup given the optimal number of replications which is explained further in this section.

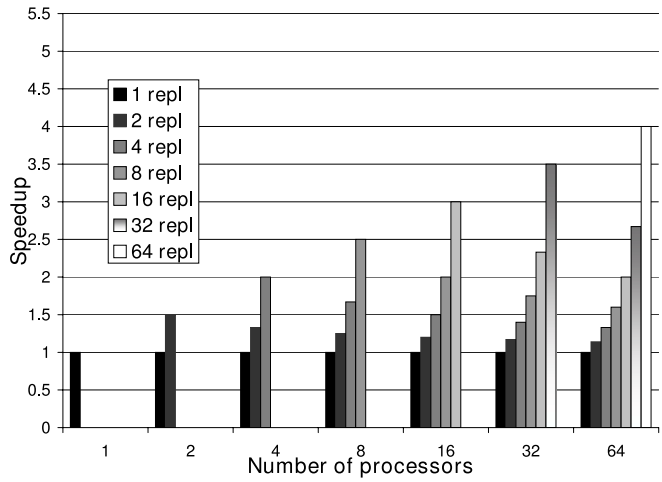


Figure 5.5: Speedup of 2^r-replications on 2^p processors, $c_X = 1.00$

Next, we illustrate examples of speedup figures for $c_X = 1.00$ and $c_X = 0.95$ that can be derived from the above formulas. With those formulas it is possible to derive speedups or iteration times for many other situations as presented here. Figure 5.5 depicts the speedup, as defined by Speedup $R(t, R, P)$ in Section 2.5, against the number of processors, $P = 2^r$, $p = 1, 2, \ldots, 6$ for different numbers of replications, $R = 2^r$, $r = 1, 2, \ldots, p$. This definition of the speedup is a comparison of JR to ELB with the same amount of processors. We observe the following from the graphical representation. For the case that $c_X = 1.00$ significant speedups can be gained by JR. Moreover, for a fixed number of processors in the resource set, the speedup consistently increases if the number of replications increases. Finally, we notice that the run times decrease maximally if the number of replications equals the number of processors.

Figure 5.6 can be derived by re-shifting the data of Figure 5.5. This figure exhibits the speedups against the number of replications, $R = 2^r$, $r = 0, 1, \ldots, p$, for different numbers of processors, $P = 2^p$, $p = 1, 2, \ldots, 6$. We notice from this figure that the speedup factor for a given number of replications decreases if the number of processors increases.

As explained above in this section, model 1.1 with $c_X = 1.00$ is closely related to a model with exponentially distributed job runtimes. Figure 5.7 illustrates for the same settings as Figure 5.6 the speedups computed by a model that represents the JR run times with exponentially distributed and completely dependent job runtimes, which means that $\alpha = \frac{1}{2},\ \beta(i,j) = \frac{1}{2}$ for $i < j,\ \beta(i,j) = \sqrt{5}/2$ for $i = j$ and $\gamma = 1$. We observe that the heights of the speedups correspond to those of Figure 5.6. For higher numbers of processors (above 8) in combination with higher numbers of replications, the speedups show the most differences due to the difference between the $\beta(i,j)$ values

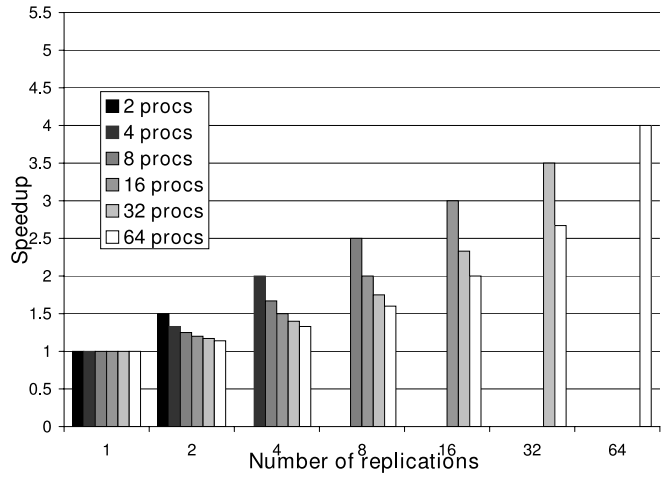


Figure 5.6: Speedup on 2^p processors of a 2^r-replication run, $c_X = 1.00$

for $i < j$, which are often applied in the approximations for those type of cases.

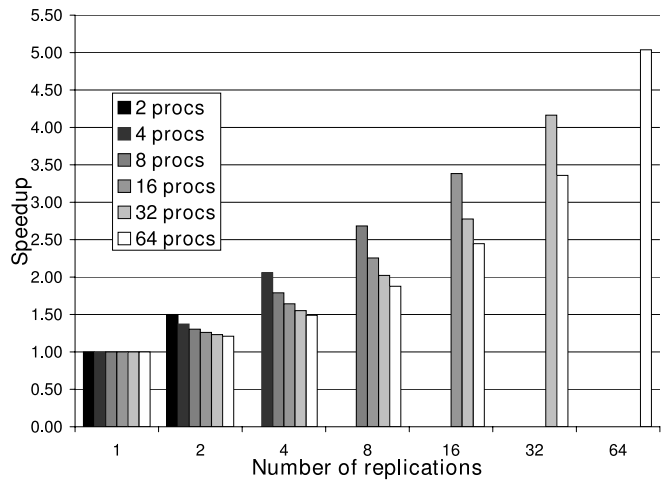


Figure 5.7: Speedup on 2^p processors of a 2^r-replication run with exponentially distributed job runtimes

Due to the reason that the speedups presented in Figures 5.5 and 5.6 are comparisons between JR and ELB with the same amount of processors it is hard to compare the speedups for two settings with different numbers of processors. In order to confront this, Figure 5.8 depicts for the same R,P-combinations the speedups of JR-runs compared to the duration of a run on one processor, which is by definition an ELB run. This figure shows that the speedup consistently increases if the number of processors or the number of replications increases. It seem that there does not exist a number of processors for which holds that the optimal number of replications does not equal P. We get into more detail further in this section.

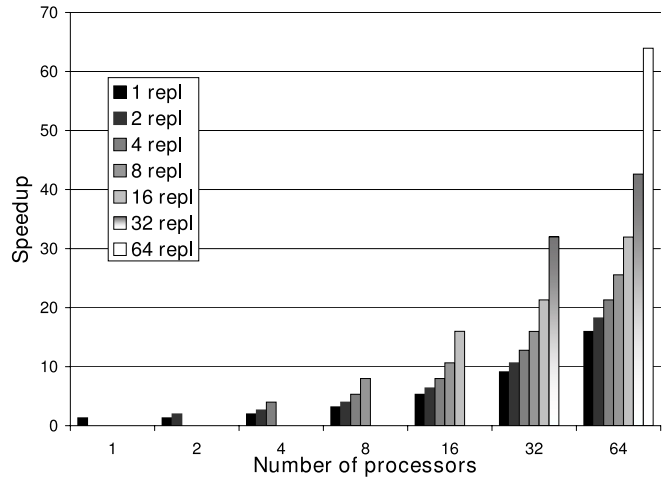


Figure 5.8: Speedup on 2^p processors of a 2^r-replication run, $c_X = 1.00$

Contrary to the figures above, the next figure illustrates that for $c_X = 0.95$ the optimal number of replications does not equal the number of processors for all settings. For $c_X = 0.95$, and for $P = 2^p$,$p = 1, 2, \ldots, 6$, Figure 5.9 depicts the speedups corresponding to different numbers of replications, $R = 2^R$, $r = 0, 1, \ldots, p$. This figure leads to the following observations about the JR speedups if $c_X = 0.95$. If the number of processors is 16 or higher, the optimal number of replications equals 16, for lower numbers the optimal number equals the number of processors. By definition the P acts as the upper bound of the optimal number of replications. Given that the P is high enough, the optimal number of replications in the analyzed figures seems to depend only on the c_X: the higher the c_X the higher the optimal number of replications. In order to investigate this in general it is necessary to derive a formula for the optimal number of replications.

To this end, we compute the derivative of (5.35) to r, equate it with 0 and round it off to derive an estimated optimal r, r^*, for $\frac{\ln(2)}{\ln(2)+\frac{1}{2}} < c_X < \frac{2^{1+p}\ln(2)}{1+2^{1+p}\ln(2)}$:

$$r^* \approx \lfloor \frac{\ln\left(\frac{c_X}{1-c_X}\right) - \ln(2) - \ln(\ln(2))}{\ln(2)} + \frac{1}{2} \rfloor \approx \lfloor 1.44 \ln\left(\frac{c_X}{1-c_X}\right) - 0.22 \rfloor. \quad (5.37)$$

Consequently, the optimal number of replications for $0 \leq c_X \leq 1$, $P = 2^p$, $p = 0, 1, \ldots$ is

$$r^* \approx \begin{cases} 0, & \text{if } 0 < c_X \leq \frac{\ln(2)}{\ln(2)+\frac{1}{2}} \approx 0.58, \\ p, & \text{if } \frac{2^{1+p}\ln(2)}{1+2^{1+p}\ln(2)} \approx \frac{1.39*2^p}{1+1.39*2^p} < c_X \leq 1, \\ \lfloor 1.44 \ln\left(\frac{c_X}{1-c_X}\right) - 0.22 \rfloor, & \text{otherwise.} \end{cases} \quad (5.38)$$

We observe from (5.38) that if the c_X is between $\ln(2)/\ln(2) + \frac{1}{2}$ and $2^{1+p}\ln(2)/1 + 2^{1+p}\ln(2)$ that the r^* is independent of the number of processors P

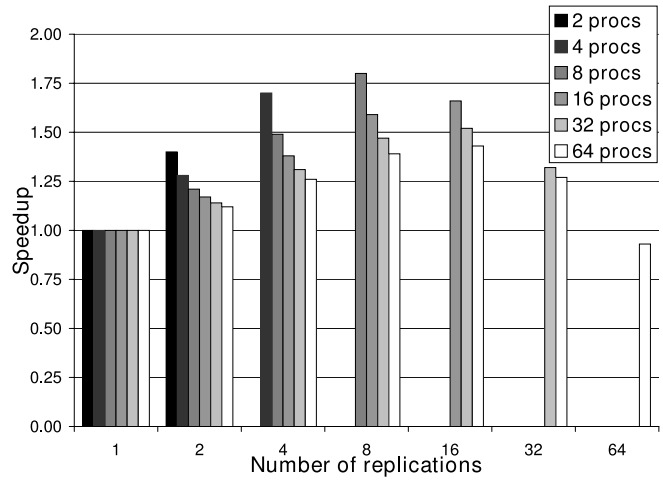


Figure 5.9: Speedup of a 2^r-replication run on 2^p processors, $c_X = 0.95$

and is an increasing function of c_X. In the next section about the computations with model 1.2 we analyze the dependency between the r^* and the other parameters on the basis of a graphical representation. The representation and consequently the observations are representative for this model. Moreover, from (5.38) we derive that speedup is gained if

$$\frac{4\ln(2)}{\sqrt{2}+4\ln(2)} \approx 0.66 < c_X \leq 1. \tag{5.39}$$

We derive the speedups for each $P = 2^p$, $p = 1, 2, \ldots$ and c_X $\left(\frac{\ln(2)}{\ln(2)+\frac{1}{2}} < c_X < \frac{2^{1+p}\ln(2)}{1+2^{1+p}\ln(2)}\right)$ by substituting r^* in (5.35):

$$\begin{aligned} \text{Speedup } R(t, R^*, P) &= \text{Speedup } R(t, 2^{r^*}, 2^p) \qquad (5.40)\\ &\approx \frac{(2+pc_X)\ln(2)}{c_X\left(3\ln(2)+1+p\ln(2)+\ln(\ln(2))+\ln\left(\frac{1-c_X}{c_X}\right)\right)}. \end{aligned}$$

Moreover, for $0 \leq c_X < \frac{\ln(2)}{\ln(2)+\frac{1}{2}}$:

$$\text{Speedup } R(t, R^*, P) \approx \text{Speedup } R(t, 1, P) \approx 1. \tag{5.41}$$

Furthermore, for $\frac{2^{1+p}\ln(2)}{1+2^{1+p}\ln(2)} < c_X \leq 1$:

$$\text{Speedup } R(t, R^*, P) \approx \text{Speedup } R(t, P, P) \approx \frac{1+\frac{1}{2}pc_X}{(1-c_X)2^p + c_X}. \tag{5.42}$$

In the next section a graphical representation of the speedup with r^* against c_X for different values for p is shown.

5.4.2 Model 1.2

In this section, we derive formulas for the iteration times of replicating $R = 2^r,\ r = 0, 1, \ldots, p$ times on $P = 2^p,\ p = 1, 2, \ldots$ processors on the basis of model 1.2, which is described in Table 5.7. In addition, we obtain formulas for the speedup with input parameters r, p, α, β_1, β_2, and c_X. Those parameters together represent the key properties of the experimental setting. Furthermore, we deduce the optimal number of replications as a function of α and c_X for $\beta_2 = \frac{1}{2}$, and moreover, the corresponding speedup given that $\beta_1 = 1$. We analyze the resulting formulas on the basis of graphical representations. We notice that all equations in this model hold for problem size $t > 0$, which is described in Section 2.5 and for $\sigma(X) \leq \mathbb{E}(X)$ (i.e., $0 \leq c_X \leq 1$) according to the domain of the approximations, defined in Section 5.3.

Given the model described by (5.6), the approximations in Section 5.3, and the details of model 1.2 in Table 5.7, we derive the following iteration times for $0 \leq \alpha \leq 1$, $R = 2^r, r = 0, 1, \ldots, p$, $P = 2^p$, $p = 0, 1, \ldots$, $\beta_1, \beta_2 \geq 0$ and $\beta_1, \beta_2 \neq 1$:

$$R_{it}(t, R, P) = R_{it}(t, 2^r, 2^p) \approx 2^r\mathbb{E}(X) - 2^r\alpha\frac{1-\beta_2^r}{1-\beta_2}\sigma(X) + 2^r\alpha\beta_2^r\frac{1-\beta_1^{p-r}}{1-\beta_1}\sigma(X), \tag{5.43}$$

which leads to the following speedup for the same set of parameter values:

$$\text{Speedup } R(t, R, P) = \text{Speedup } R(t, 2^r, 2^p) \approx \frac{1 + \alpha c_X \frac{1-\beta_1^p}{1-\beta_1}}{2^r + 2^r\alpha c_X(\beta_2^r\frac{1-\beta_1^{p-r}}{1-\beta_1} - \frac{1-\beta_2^r}{1-\beta_2})}, \tag{5.44}$$

For $0 \leq \alpha \leq 1$, $R = 2^r$, $r = 0, 1, \ldots, p$, $P = 2^p$, $p = 0, 1, \ldots$, $\beta_1 = 1, \beta_2 \geq 0$ and $\beta_2 \neq 1$:

$$R_{it}(t, R, P) = R_{it}(t, 2^r, 2^p) \approx 2^r\mathbb{E}(X) - 2^r\alpha\frac{1-\beta_2^r}{1-\beta_2}\sigma(X) + 2^r\alpha\beta_2^r(p-r)\sigma(X). \tag{5.45}$$

Consequently, the speedup for the same set of parameters is

$$\text{Speedup } R(t, R, P) = \text{Speedup } R(t, 2^r, 2^p) \approx \frac{1 + \alpha p c_X}{2^r + 2^r\alpha c_X(\beta_2^r(p-r) - \frac{1-\beta_2^r}{1-\beta_2})}. \tag{5.46}$$

In some cases these speedup formulas derive speedups that are lower than 0 or go to infinity. We expect that for these settings JR leads to a high speedup (> 5). The formula does not provide accurate estimations for those cases. Unfortunately, it is not possible to derive easy equations that illustrate for which parameters the speedup has a value between 0 and infinity.

To derive an optimal r^* we need to take the derivative to r of this formula. That derivative only exists if α, β_1, β_2 ($\neq \frac{1}{2}$), p, and c_X are known or if $\beta_2 = \frac{1}{2}$. Below we provide the solution of the r^* for $\beta_2 = \frac{1}{2}$. To this end, we compute the derivatives of (5.44) and (5.46) to r and equalize it with 0 to derive r^* for $p = 0, 1, \ldots$, and $\beta_2 = \frac{1}{2}$, for $\beta_1 \neq 1$, and for $\beta_1 = 1$. Given $\beta_1 \neq 1$, the optimal number of replications before rounding off equals

$$r^* \approx \frac{\ln(\beta_1)p + \ln(\frac{\ln(\beta_1)}{\beta_1 - 1}) + \ln(\alpha) + \ln(\frac{c_X}{1-2\alpha c_X}) - \ln(\ln(2))}{\ln(2\beta_1)}, \text{ with } \beta_1 = \sqrt{1\frac{3}{4} - 2\alpha^2}. \tag{5.47}$$

According to Approximation 5.3.7 a value of $\alpha = \frac{1}{4}\sqrt{6} \approx 0.61$ would lead to a β_1 value of 1, given that $\beta_2 = \sqrt{2 - 2\alpha^2 - \beta_1^2} = \frac{1}{2}$. As a result, for $\alpha = \frac{1}{4}\sqrt{6}$, $\beta_1 = 1$, $\beta_2 = \frac{1}{2}$, and $p = 0, 1, \ldots$, the optimal number of replications equals

$$r^* \approx \begin{cases} 0, & \text{if } \frac{\alpha c_X}{1-2\alpha c_X} \leq \ln(2), \\ p, & \text{if } \frac{\alpha c_X}{\ln(2)-2\ln(2)\alpha c_X} \geq 2^p, \\ \lfloor \frac{\ln(\alpha)+\ln\left(\frac{c_X}{1-2\alpha c_X}\right)-\ln(\ln(2))}{\ln(2)} + \frac{1}{2} \rfloor, & \text{otherwise.} \end{cases} \tag{5.48}$$

We conclude from this equation that if $\alpha = \frac{1}{4}\sqrt{6}$, $\beta_1 = 1$, and $\beta_2 = \frac{1}{2}$, and there are enough processors in the resource set, denoted by

$$p > \frac{1}{\ln(2)} \ln\left(\frac{c_X}{\frac{4\ln(2)}{\sqrt{6}} - 2\ln(2)c_X}\right) \approx 1.44 \ln(c_X/(1.13 - 0.69c_X)), \tag{5.49}$$

the optimal number of replications does not depend on p.

To gain insight in the impact of the α, β_1, and c_X on the optimal number of replications given that $\beta_2 = \frac{1}{2}$, we construct Figure 5.10 which depicts the r^* against different values of α, β_1, and β_2 and observe the following. This figure clearly shows the positive relation between the r^* and the c_X which is stated above. Every r^* line seem to have an asymptote and the c_X value at which there is an asymptote differs. The higher the α with a corresponding lower β_1, the lower the optimal number of replications, r^*, for lower c_X (below 0.75) and the higher the r^* for c_X higher than 0.75. Moreover, for those αs the value of the c_X at which there is an asymptote is lower.

The speedup formula, given that $\beta_1 \neq 1, \beta_2 = \frac{1}{2}$ can be derived by substituting (5.47) in (5.44). Due to the fact that this formula is too long we do not write it down. Substituting r^* from (5.48) in the speedup formula (5.46) for $\beta_1 = 1$, and $\beta_2 = \frac{1}{2}$, leads to a shorter and writable speedup-estimation formula. We derive the following formula given the optimal number of replications for $\alpha = \frac{1}{4}\sqrt{6}$, $\beta_1 = 1$, $\beta_2 = \frac{1}{2}$, $p = 0, 1, \ldots$:

$$\text{Speedup } R(t, R^*, P) = \text{Speedup } R(t, 2^{r^*}, 2^p) \tag{5.50}$$

$$\approx \begin{cases} 1, & \text{if } \frac{\alpha c_X}{1-2\alpha c_X} \leq \ln(2), \\ \frac{1+\alpha p c_X}{2^p - \alpha c_X(2^{p+1}-2)}, & \text{if } \frac{\alpha c_X}{\ln(2)-2\ln(2)\alpha c_X} \geq 2^p, \\ \frac{\ln(2)+\alpha\ln(2)c_X p}{\alpha c_X\left(1+3\ln(2)+\ln(\ln(2))+\ln(2)p+\ln\left(\frac{1-2\alpha c_X}{2\alpha c_X}\right)\right)}, & \text{otherwise.} \end{cases}$$

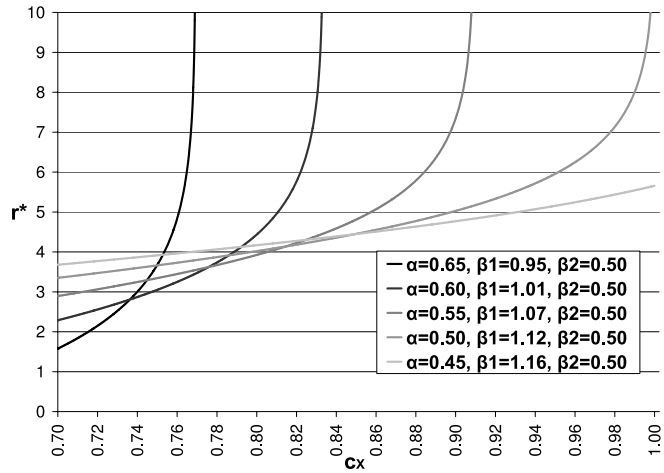


Figure 5.10: r^* against the c_X for $\beta_2 = \frac{1}{2}$, and different α and β_1

Consequently, we are able to generate Figure 5.11 which represents the highest possible speedup against the p for $c_X = 0.65, 0.70, 0.75, 0.80, \frac{\sqrt{6}}{3}$. We conclude that for the case that $c_X = \frac{1}{3}\sqrt{6}$ the speedup grows linearly with the number of processors. A higher c_X that is not represented by the model, will lead to even higher speedups. If $c_X < \frac{1}{3}\sqrt{6}$ the speedup first grows by adding more processors to the resource set, and finally decreases after some number of processors. This number of processors is higher if the c_X is higher and can be derived with (5.50). We conclude that a small increase in the c_X can have a high impact on the speedup.

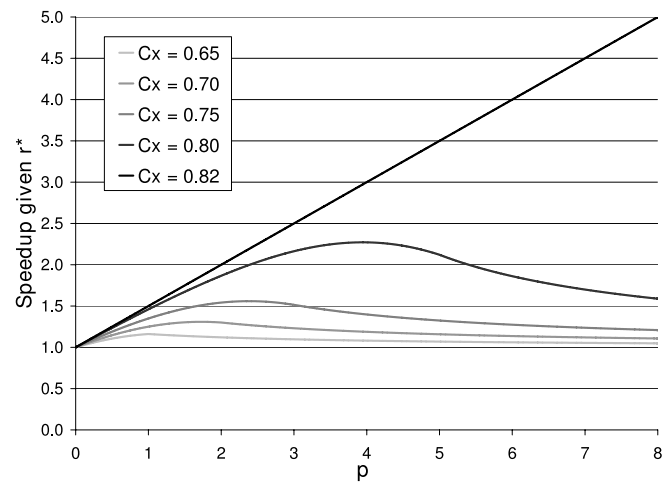


Figure 5.11: Speedup versus p, for $c_X = 0.65, 0.70, 0.75, 0.80, \frac{1}{3}\sqrt{6}$, given r^*

In addition, Figure 5.12 provides a graphical representation of the speedups that can be gained by JR according to model 1.1 and 1.2 for $c_X, 0.70 \leq c_X \leq 1.00$, for different numbers of processors, $p = 2, 5, 10, 20, 100$, given that for every case the optimal number of replications. As can be seen in the representation there is an asymptote in the speedup at $c_X = 1.00$ for all of the different numbers of p. This behavior can moreover be derived from (5.40). For model 1.2 we observe in this figure that there is an asymptote for a lower value of c_X. With (5.50) we derive that this asymptote is at $c_X = 1/3\sqrt{6} \approx 0.82$ for $p = 1, 2, \ldots$.

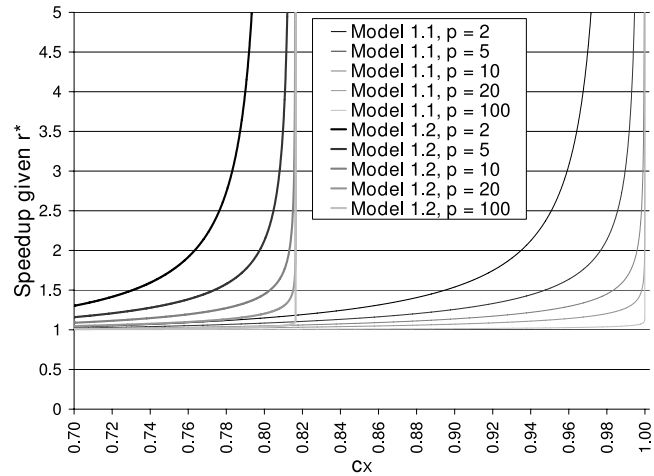


Figure 5.12: Speedup versus c_X, given r^*

The graphical representation give insight in the properties of the speedups. In addition, by combining the representations we get more understanding about their characteristics. For instance, we observe in Figure 5.10 that the optimal number of replications equals 2 if $c_X = 0.70$, $\alpha = 0.60$, $\beta_1 = 1.01$, and $\beta_2 = 0.50$. For a JR setting with 2^{10} processors, these parameters can be substituted in (5.44), and we derive that JR leads to a speedup of 1.09. The same speedup can be derived in Figure 5.12 by reading off the speedup that corresponds to model 1.2 with $p = 10$, and $c_X = 0.70$. Moreover, we notice in Figure 5.10 that the r^* equals the p if the c_X increases to above 0.82. In (5.43), we observe that for this case the expected iteration time is much lower than for a run without replicating. Consequently, this means that the speedup is high, which corresponds to what has been presented by Figure 5.12.

5.4.3 Model 1.3

In this section, we derive formulas for the iteration times of replicating $R = 2^r$, $r = 0, 1, \ldots, p$ times on $P = 2^p$, $p = 0, 1, \ldots$ processors on the basis of model 1.3, which is described in Table 5.7. In addition, we obtain formulas for the speedup with input parameters r, p, α, $\beta(i, j)$, $i = 0, 1, \ldots, p$, $j = 1, 2, \ldots, p$, and c_X. Those parameters together represent the key properties of the experimental setting. We notice that all equations in this model hold for problem size $t > 0$, which is described in

Section 2.5 and for $\sigma(X) \leq \mathbb{E}(X)$ (i.e., $0 \leq c_X \leq 1$) according to the domain of the approximations, defined in Section 5.3.

Given the model described by (5.6), the approximations in Section 5.3, and given the properties of α, and $\beta(i,j)$, as stated in Table 5.7, we derive the following iteration times.

For replicating 2^r, $r = 0$, times on $P = 2^p$, $p = 0, 1, \ldots$ processors with $0 \leq \alpha \leq 1$, $\beta(i,j) \geq 0$ for $i = 0, 1, \ldots, p$ and $j = 0, 1, \ldots, p$ the iteration time estimation is

$$R_{it}(t,1,P) = R_{it}(t,2^0,2^p) \approx \mathbb{E}(X) + \alpha\sigma(X)\left(1 + \sum_{i=r+1}^{p-1}\prod_{j=r+1}^{i}\beta(r,j)\right), \quad (5.51)$$

Moreover, for replicating 2^r, $r = 0, 1, \ldots, p$, times on $P = 2^p$, $p = 0, 1, \ldots$ processors with $0 \leq \alpha \leq 1$, and $\beta(i,j) \geq 0$ for $i = 0, 1, \ldots, r$ and $j = 0, 1, \ldots, p$ the iteration time estimation equals

$$\begin{aligned} R_{it}(t,R,P) = R_{it}(t,2^r,2^p) \approx 2^r\mathbb{E}(X) - 2^r\alpha\sigma(X)\left(1 + \sum_{i=1}^{r-1}\prod_{k=1}^{i}\beta(k,k)\right) \\ + 2^r\alpha\sigma(X)\prod_{i=1}^{r}\beta(i,i)\left(1 + \sum_{i=r+1}^{p-1}\prod_{j=r+1}^{i}\beta(r,j)\right), \end{aligned} \quad (5.52)$$

and given that $\beta(i,j) = 1$ for $i = 0, 1, \ldots, r$, $j = 0, 1, \ldots, p$ where $i < j$ the iteration time estimation is

$$\begin{aligned} R_{it}(t,R,P) = R_{it}(t,2^r,2^p) \approx 2^r\mathbb{E}(X) - 2^r\alpha\sigma(X)\frac{1 - (\sqrt{1-2\alpha^2})^r}{1 - \sqrt{1-2\alpha^2}} \\ + 2^r\alpha\left(\sqrt{1-2\alpha^2}\right)^r(p-r)\sigma(X). \end{aligned} \quad (5.53)$$

These iteration times lead to the following speedups. For replicating 2^r, $r = 0, 1, \ldots, p$, times on $P = 2^p$, $p = 0, 1, \ldots$ processors with $0 \leq \alpha \leq 1$, and $\beta(i,j) \geq 0$ for $i = 0, 1, \ldots, r$ and $j = 0, 1, \ldots, p$ the speedup estimation equals

$$\begin{aligned} &\text{Speedup } R(t,R,P) = \text{Speedup } R(t,2^r,2^p) \\ &\approx \frac{1 + \alpha c_X \prod_{i=1}^{r}\beta(i,i)\left(1 + \sum_{i=r+1}^{p-1}\prod_{j=r+1}^{i}\beta(r,j)\right)}{2^r + 2^r\alpha c_X\left(\prod_{i=1}^{r}\beta(i,i)\left(1 + \sum_{i=r+1}^{p-1}\prod_{j=r+1}^{i}\beta(r,j)\right) - \sum_{i=1}^{r-1}\prod_{k=1}^{i}\beta(k,k) - 1\right)}, \end{aligned} \quad (5.54)$$

and given that $\beta(i,j) = 1$ for $i = 0, 1, \ldots, r$, $j = 0, 1, \ldots, p$ where $i < j$ the speedup estimation equals

$$\begin{aligned} &\text{Speedup } R(t,R,P) = \text{Speedup } R(t,2^r,2^p) \\ &\approx \frac{1 + \alpha p c_X\sqrt{1-2\alpha^2}}{2^r + 2^r\alpha c_X\left(\left(\sqrt{1-2\alpha^2}\right)^r(p-r) - \frac{1-(\sqrt{1-2\alpha^2})^r}{1-\sqrt{1-2\alpha^2}}\right)}. \end{aligned} \quad (5.55)$$

In some cases these speedup formulas derive speedups that are lower than 0 or go to infinity. We expect that for these settings JR leads to a high speedup (> 5). The formula does not provide accurate estimations for those cases. Unfortunately, it is not possible to derive easy equations that illustrate for which parameters the speedup has a value between 0 and infinity. We show in Section 5.5 a graphical representation of the speedups derived by this formula.

In order to derive an optimal r^* we need to take the derivative to r of (5.54) and equate it with 0. Unfortunately, the general derivative of (5.54) does not exist and therefore it is not possible to derive a general formula for the optimal number of replications. Although the derivative of (5.55) to r does exist, equating it with 0 leads to a very long formula that contains a $LambertW$ equation, which is defined in the next section. Due to that reason, we do not present the formula in this thesis.

5.4.4 Model 2.1

In this section, we derive the speedups of replicating on homogeneous nodes. We derive formulas for the iteration times of replicating $R = 1, 2$ times on 2 processors, $R = 1, 2, 4$ times on 4 processors, and $R = 2^r$, $r = 0, 1, \ldots, p$ times on $P = 2^p$, $p = 0, 1, \ldots$ processors on the basis of model 2.1, which is described in Table 5.7. In addition, we obtain formulas for the speedup with input parameters R, P, and c_X. Those parameters together represent the key properties of the experimental setting. Furthermore, we deduce the optimal number of replications as a function of P and c_X. We analyze the resulting formulas on the basis of graphical representations. We notice that all equations in this model hold for problem size $t > 0$, which is described in Section 2.5 and for $\sigma(X) \leq \mathbb{E}(X)$ (i.e., $0 \leq c_X \leq 1$) according to the domain of the approximations, defined in Section 5.3.

Unlike the sections about models 1.1-1.3, we do not make the assumption that the standard deviation of the sum of stochastic variables equals the sum of the standard deviations. Note that the duration of the iteration time for $R = 1$ and for $R = P$ does not depend on that assumption; $R_{it}(t, 1, P)$ and $R_{it}(t, P, P)$ from the previous section are the same in model 2.1. For example, the iteration time of a two-replication run on two processors also equals $2\mathbb{E}\min\{X_{111}, X_{211}\}$:

$$\begin{aligned} R_{it}(t, 2, 2) = R_{it}(t, 2^1, 2^1) &= \mathbb{E}\sum_{j=1}^{2} \min_{k=1,2}\{X_{1jk}\} \\ &= \sum_{j=1}^{2} \mathbb{E}\min_{k=1,2}\{X_{1jk}\} = 2\mathbb{E}\min\{X_{111}, X_{211}\}. \end{aligned} \tag{5.56}$$

However, the speedups for $1 < R < P$ are different to those of the previous section.

Replicating on two processors

As explained above, the results for replicating on two processors are the same as in the previous section and therefore we only shortly present the final results in this section. For the computation details we refer to model 1.1. The expected iteration times of replicating are

$$R_{it}(t,1,2) \approx \mathbb{E}(X) + \frac{1}{2}\sigma(X), \tag{5.57}$$

and

$$R_{it}(t,2,2) \approx 2\mathbb{E}(X) - \sigma(X). \tag{5.58}$$

The optimal number of replications for $c_X \geq 0$ is

$$R^* \approx \begin{cases} 1, & \text{if } 0 \leq c_X \leq \frac{2}{3}, \\ 2, & \text{if } \frac{2}{3} < c_X \leq 1. \end{cases} \tag{5.59}$$

The speedup of a 2-JR run equals

$$\text{Speedup } R(t,2,2) \approx \frac{2+c_X}{4-2c_X}. \tag{5.60}$$

Replicating on four processors

The following expected iteration time for ELB is known from model 1.1:

$$R_{it}(t,1,4) = R_{it}(t,2^0,2^2) \approx \mathbb{E}(X) + \sigma(X). \tag{5.61}$$

For the case of $R = 2$ replications, we derive the following

$$\begin{aligned}
&R_{it}(t,2,4) = R_{it}(t,2^1,2^2) = V_{1,2}(0) = \mathbb{E}\max\{V_{1,1}(0), V_{1,1}(1)\} \\
&\approx \mathbb{E}V_{1,1}(0) + \frac{1}{2}\sigma(V_{1,1}(0)) \\
&= \mathbb{E}\sum_{m=0,1}\min\{V_{0,0}(2m), V_{0,0}(2m+1)\} \\
&\qquad + \frac{1}{2}\sigma(\sum_{m=0,1}\min\{V_{0,0}(2m), V_{0,0}(2m+1)\}) \\
&\approx 2\mathbb{E}\min\{V_{0,0}(0), V_{0,0}(1)\} + \frac{1}{2}2^{1\times\gamma(=\frac{1}{2})}\sigma(\min\{V_{0,0}(0), V_{0,0}(1)\}) \\
&\approx 2\mathbb{E}(V_{0,0}(0)) - 2\times\frac{1}{2}\sigma(V_{0,0}(0)) + \frac{1}{2}\sqrt{2}\frac{1}{2}\sigma(V_{0,0}(0)) \\
&= 2\mathbb{E}(X) + (\frac{1}{4}\sqrt{2} - 1)\sigma(X).
\end{aligned} \tag{5.62}$$

The first step is trivial, the second applies (5.1), the third uses (5.6), the fourth uses Approximation 5.3.3, the fifth applies (5.6), the sixth step uses Approximation 5.3.5, the seventh Approximations 5.3.6 and 5.3.7, and the last step is according to (5.5) and some trivial computations.

For the case of $R = 4$ replications, we know from model 1.1 that the expected iteration time is

$$R_{it}(t, 4, 4) = R_{it}(t, 2^2, 2^2) \approx 4\mathbb{E}(X) - 3\sigma(X). \tag{5.63}$$

The speedup for replicating $R = 2$ times for $c_X < \frac{8}{4-\sqrt{2}}$ (≈ 3.09) can be obtained by taking (5.61) over (5.62). We derive:

$$\text{Speedup } R(t, 2, 4) = \text{Speedup } R(t, 2^1, 2^2) \approx \frac{4 + 4c_X}{8 + \sqrt{2}c_X - 4c_X}. \tag{5.64}$$

As a consequence, we are able to compute for which c_X a 2-JR setting leads to a speedup in comparison with no replication. That is the case if (5.64) is higher than 1, which can be rewritten to:

$$c_X > \frac{4}{8 - \sqrt{2}} \approx 0.61. \tag{5.65}$$

Next, we present the speedup for 4-JR which is the same as the speedup of 4-JR in model 1.1. The speedup for this setting given $c_X < \frac{4}{3}$ equals

$$\text{Speedup } R(t, 4, 4) = \text{Speedup } R(t, 2^2, 2^2) \approx \frac{1 + c_X}{4 - 3c_X}. \tag{5.66}$$

For $c_X \geq \frac{4}{3}$ we expect that the approximations are inaccurate and therefore further analyses are needed to investigate the speedups for these cases. We perform no investigations into those issues because of the fact that it is beyond the scope of this thesis. A comparison of the 2-JR and the 4-JR speedups illustrates that $R = 4$ leads to a higher speedup than $R = 2$ if:

$$c_X > \frac{8}{\sqrt{2} + 8}. \tag{5.67}$$

Consequently, combining the above comparisons, we advice the following numbers of replications for a given $c_X \geq 0$ is

$$R^* \approx \begin{cases} 1, & \text{if} \quad 0 & \leq c_X & \leq \frac{4}{8-\sqrt{2}} \\ 2, & \text{if} \quad \frac{4}{8-\sqrt{2}} & < c_X & \leq \frac{8}{\sqrt{2}+8} \\ 4, & \text{if} \quad \frac{8}{\sqrt{2}+8} & < c_X & \leq 1. \end{cases} \tag{5.68}$$

Replicating $R = 2^r$ times on $P = 2^p$ processors

In this section, we derive formulas for the iteration times and speedups for $R = 2^r$, $r = 0, 1, \ldots, p$, JR on $P = 2^p$, $p = 0, 1, \ldots$, processors and fixed values of parameters α, $\beta(i, j)$ and γ, which are specified in Table 5.7. For $R = 1$, and $R = P$, the formulas of model 2.1 are the same as for model 1.1. In order to obtain the formulas for the iteration times and the speedups given $1 < R < P$ we extend the computations for the settings with 2 and 4 processors. For the complete derivations of the formulas we refer to the previous sections.

As stated above, the iteration time of the non-replication case for $p = 0, 1, \ldots$ equals those of model 1.1 which is

$$R_{it}(t, 1, P) = R_{it}(t, 2^0, 2^p) \approx \mathbb{E}(X) + \frac{1}{2} p\sigma(X). \tag{5.69}$$

For $P = 2^p$, $p = 0, 1, \ldots$, processors, and $R = 2^r$, $r = 0, 1, \ldots, p$, replications, the expected iteration time equals

$$R_{it}(t, R, P) = R_{it}(t, 2^r, 2^p) \approx 2^r \mathbb{E}(X) + \left(1 - 2^r + \frac{p - r}{2^{\frac{1}{2}r+1}}\right) \sigma(X). \tag{5.70}$$

We know from the previous section that the iteration time of an $R = P$-replication run on $P = 2^p$ processors for $p = 0, 1, \ldots$ is

$$R_{it}(t, P, P) = R_{it}(t, 2^p, 2^p) \approx 2^p \mathbb{E}(X) - (2^p - 1)\sigma(X). \tag{5.71}$$

Formula (5.69) over (5.70) represents the speedup that can be gained by replication for $r = 0, 1, \ldots, p$, $p = 0, 1, \ldots$. We derive:

$$\begin{aligned} \text{Speedup } R(t, R, P) &= \text{Speedup } R(t, 2^r, 2^p) \\ &\approx \frac{2 + pc_X}{2^{r+1} + c_X \left(2 - 2^{r+1} + (p - r) * \left(\frac{1}{2}\sqrt{2}\right)^r\right)}. \end{aligned} \tag{5.72}$$

This formula derives for every c_X, $0 \leq c_X \leq 1$, and the given values for r and p a speedup value between 0 and infinity.

To make a comparison between Models 1.1 and 2.1, we demonstrate a speedup figure for setting with $P = 2^p, p = 0, \ldots, 6$ processors and $R = 2^r, r = 0, \ldots, p$ replications with $c_X = 0.95$. Figure 5.13 depicts those speedups against the parameter r. We notice from this figure that the assumption that $\gamma = \frac{1}{2}$ (this model) instead of $\gamma = 1$ (model 1.1) leads for many settings to a higher approximation of the speedup. This means that the lower the dependency between consecutive jobs the higher the speedup that can be gained by JR.

The speedup of a 2^p-replication run on 2^p processors equals the following formula that corresponds to those of model 1.1:

$$\text{Speedup } R(t, P, P) = \text{Speedup } R(t, 2^p, 2^p) \approx \frac{1 + \frac{1}{2} pc_X}{(1 - c_X)2^p + c_X}. \tag{5.73}$$

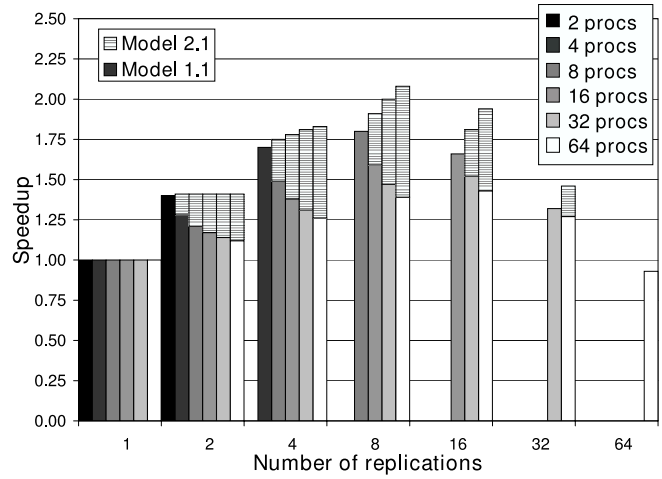


Figure 5.13: Speedup of a 2^r-replication run on 2^p processors, $c_X = 0.95$

We need the following definition of the $LambertW$ equation for further investigations.

Definition 5.4.1: Given W as any complex number and e^W as an exponential function, the LambertW-function is defined as the inverse function, $f^{-1}(W)$, of

$$f(W) = We^W.$$

More information about the Lambert-function can be found in [49].

We compute the derivative of (5.72) to r and equate it with 0 to derive the formula for the optimal r, r^* for $p = 0, 1, \ldots$:

$$r^* \approx \begin{cases} 0, & \text{if } 0 \leq c_X \leq \frac{4\ln(2)}{4\ln(2)+2+\ln(2)p} \approx \frac{2.8}{4.7+0.7p}, \\ p, & \text{if } \frac{1}{1+\frac{1}{2^{1+\frac{3}{2}p\ln(2)}}} \approx \frac{1}{1+\frac{1}{1.4*2^{1+1.5p}}} \leq c_X \leq 1, \\ \lfloor \frac{6+3\ln(2)p-2LambertW(-6\frac{\ln(2)(-1+c_X)e^{3+\frac{3}{2}\ln(2)p}}{c_X})}{3\ln(2)} + \frac{1}{2} \rfloor, & \text{otherwise.} \end{cases} \tag{5.74}$$

Formula (5.74) illustrates that the optimal number of replications is a function of c_X and p. Due to the $LambertW$ part it is not easy to observe the characteristics of this function. For that reason, Figure 5.14 illustrates the r^* against the 2-log of the number of processors, $p = 0, 1, \ldots, 7$, for $c_X = 0.42, 0.70, 0.90, 0.97, 1.00$. We observe that even if $c_X = 0.42$, which is a very low coefficient of variation, the optimal number of replications equals 1 if $p = 7$. This indicates that even for this situation, JR leads to speedups compared to ELB. Moreover, this figure illustrates that if the c_X increases the r^* increases. In addition, we observe that the r^* depends on the p, which can be seen in the jumps of r^* to higher levels if the number of processors in the resource set is

higher. A c_X of 1.0 leads to an optimal number of replications that equals the number of processors. This observation corresponds to [47]. As can be seen in (5.38), the r^* of Model 1.1 differs from Model 2.1 in the dependency on p. Model 1.1 shows that r^* does not depend on p and a plot of the r^* in that model would show that the r^*s with different c_X would increase with the p until the maximum of r^* is reached; for higher p the r^* stays constant at that level.

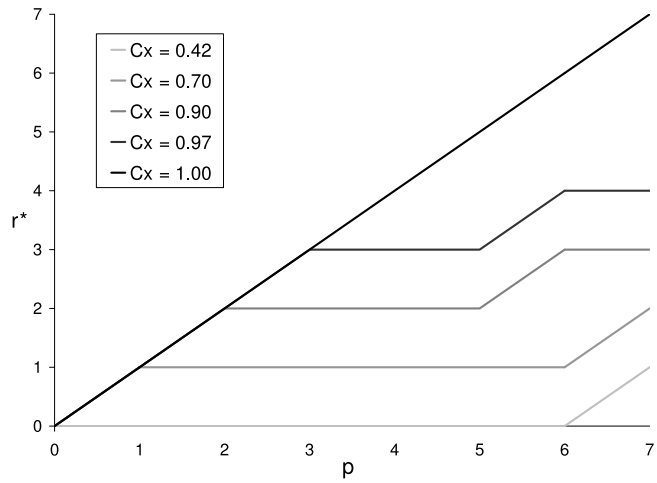


Figure 5.14: r^* versus p, for $c_X = 0.42, 0.70, 0.90, 0.97, 1.0$

Given the r^*, the speedups can be derived for each p and c_X by substituting (5.74) in (5.72). This formula for r^* is too long and therefore we do not present it in this thesis. However, the speedup graph against the 2-log of the number of processors for different c_X given r^* is presented. Figure 5.15 depicts those speedups for $p = 0, 1, \ldots, 7$, $c_X = 0.50, 0.65, 0.80, 0.85, 0.95, 1.00$. Observe that the speedup consistently increases with the p for all c_X. For a $c_X = 1.00$ the speedup increases linearly with the p. Moreover, notice that for a fixed number of processors the speedup increases fundamentally with the c_X.

5.4.5 Model 2.2

In this section, we derive formulas for the iteration times of replicating $R = 2^r,\ r = 0, 1, \ldots, p$ times on $P = 2^p,\ p = 0, 1, \ldots$ processors on the basis of model 1.3, which is described in Table 5.7. In addition, we obtain formulas for the speedup with input parameters r, p, α, $\beta(i, j)$, $i = 0, 1, \ldots, r$, $r = 0, 1, \ldots, p$, $j = 1, 2, \ldots, p$, $p = 0, 1, \ldots$, and $0 \leq c_X \leq 1$. Those parameters together represent the key properties of the experimental setting. We notice that all equations in this model hold for problem size $t > 0$, which is described in Section 2.5 and for $\sigma(X) \leq \mathbb{E}(X)$ (i.e., $0 \leq c_X \leq 1$) according to the domain of the approximations, defined in Section 5.3.

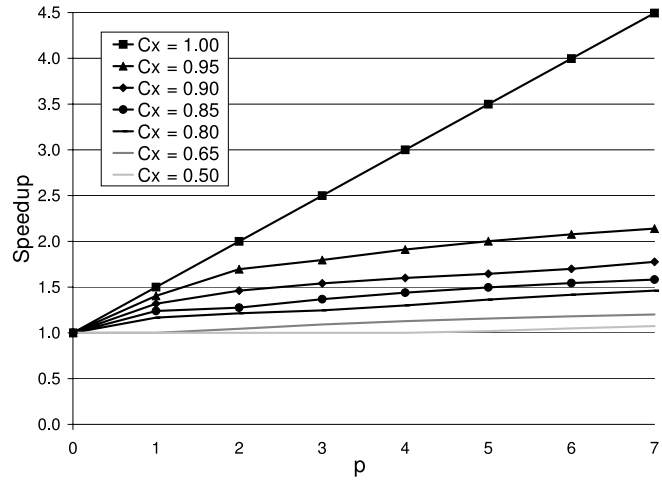


Figure 5.15: Speedup versus p, for $c_X = 0.42, 0.70, 0.90, 0.97, 1.0$, given r^*

Given the model described by (5.6), the approximations in Section 5.3, and the properties of α, β_1, β_2, and γ as presented in Table 5.7, we derive the following iteration times for $r = 0, 1, \ldots, p$, $p = 0, 1, \ldots$, and $\beta_1, \beta_2 \neq 1$:

$$R_{it}(t, R, P) = R_{it}(t, 2^r, 2^p) \approx 2^r \mathbb{E}(X) - 2^r \alpha \frac{1-\beta_2^r}{1-\beta_2} \sigma(X) + (2^r)^\gamma \alpha \beta_2^r \frac{1-\beta_1^{p-r}}{1-\beta_1} \sigma(X), \tag{5.75}$$

which leads to the following speedup for the same sets of parameters:

$$\text{Speedup } R(t, R, P) = \text{Speedup } R(t, 2^r, 2^p) \approx \frac{1 + \alpha c_X \frac{1-\beta_1^p}{1-\beta_1}}{2^r + \alpha c_X \left((2^r)^\gamma \beta_2^r \frac{1-\beta_1^{p-r}}{1-\beta_1} - 2^r \frac{1-\beta_2^r}{1-\beta_2} \right)}. \tag{5.76}$$

We derive the following estimation of the iteration time given the properties of α, β_1, β_2, γ described in Table 5.7, $\beta_1 = 1$, $\beta_2 \neq 1$, for $r = 0, 1, \ldots, p$, and $p = 0, 1, \ldots$:

$$R_{it}(t, R, P) = R_{it}(t, 2^r, 2^p) \approx 2^r \mathbb{E}(X) - 2^r \alpha \frac{1-\beta_2^r}{1-\beta_2} \sigma(X) + (2^r)^\gamma \alpha \beta_2^r (p-r) \sigma(X). \tag{5.77}$$

Consequently, the speedup for the same set of parameters is

$$\text{Speedup } R(t, R, P) = \text{Speedup } R(t, 2^r, 2^p) \approx \frac{1 + \alpha p c_X}{2^r + \alpha c_X \left((2^r)^\gamma \beta_2^r (p-r) - 2^r \frac{1-\beta_2^r}{1-\beta_2} \right)}. \tag{5.78}$$

In some cases these speedup formulas derive speedups that are lower than 0 or go to infinity. We expect that for these settings JR leads to a high speedup (> 5). The formula does not provide accurate estimations for those cases. Unfortunately, it is not possible to derive easy equations that illustrate for which parameters the speedup has a value between 0 and infinity.

To derive an optimal r^* in general we need to take the derivative of (5.76) to r and equate it with 0. Although the derivatives of (5.76) and (5.78) exist, it is not possible to equate those derivatives with 0 and derive the solution for r^*.

5.4.6 Model 2.3

In this section, we derive formulas for the iteration times of replicating $R = 2^r,\ r = 0, 1, \ldots, p$ times on $P = 2^p,\ p = 1, 2, \ldots$ processors on the basis of model 2.3, which is described in Table 5.7. In addition, we obtain formulas for the speedup with input parameters r, p, α, $\beta(i, j)$, $i = 0, 1, \ldots, p$, $j = 1, 2, \ldots, p$, and c_X. Those parameters together represent the key properties of the experimental setting. We notice that all equations in this model hold for problem size $t > 0$, which is described in Section 2.5 and for $\sigma(X) \leq \mathbb{E}(X)$ (i.e., $0 \leq c_X \leq 1$) according to the domain of the approximations, defined in Section 5.3.

Given the model described by (5.6), the approximations in Section 5.3, and the details of model 2.3 in Table 5.7, we derive the following iteration times. For an ELB run ($R = 2^r$, $r = 0$) on $P = 2^p$, $p = 0, 1, \ldots$ processors with $0 \leq \alpha \leq 1$, $\beta(i, j) \geq 0$ for $i = 0, 1, \ldots, p$ and $j = 0, 1, \ldots, p$, and $\frac{1}{2} \leq \gamma \leq 1$ we derive the following iteration time estimation:

$$R_{it}(t, 1, P) = R_{it}(t, 2^0, 2^p) \approx \mathbb{E}(X) + \alpha\sigma(X) \left(1 + \sum_{i=r+1}^{p-1} \prod_{j=r+1}^{i} \beta(r, j)\right). \quad (5.79)$$

Moreover, for replicating $2^r, r = 0, 1, \ldots, p$, times on $P = 2^p, p = 0, 1, \ldots$ processors with $0 \leq \alpha \leq 1$, $\frac{1}{2} \leq \gamma \leq 1$, and $\beta(i, j) \geq 0$ for $i = 0, 1, \ldots, r$ and $j = 0, 1, \ldots, p$ the iteration time estimation equals

$$\begin{aligned} R_{it}(t, R, P) = R_{it}(t, 2^r, 2^p) \approx 2^r\mathbb{E}(X) - 2^r\alpha\sigma(X) \left(1 + \sum_{i=1}^{r-1} \prod_{k=1}^{i} \beta(k, k)\right) \\ + 2^{r\gamma}\alpha\sigma(X) \prod_{i=1}^{r} \beta(i, i) \left(1 + \sum_{i=r+1}^{p-1} \prod_{j=r+1}^{i} \beta(r, j)\right), \end{aligned} \quad (5.80)$$

and for the same set of parameters and given that $\beta(i, j) = 1$ for $i = 0, 1, \ldots, r$, $j = 0, 1, \ldots, p$ and $i < j$:

$$\begin{aligned} R_{it}(t, R, P) = R_{it}(t, 2^r, 2^p) \approx & 2^r\mathbb{E}(X) - 2^r\alpha\sigma(X)\frac{1 - (\sqrt{1 - 2\alpha^2})^r}{1 - \sqrt{1 - 2\alpha^2}} \\ & + 2^{r\gamma}\alpha \left(\sqrt{1 - 2\alpha^2}\right)^r (p - r)\sigma(X). \end{aligned} \quad (5.81)$$

These iteration times lead to the following speedups. For replicating 2^r, $r = 0, 1, \ldots, p$, times on $P = 2^p$, $p = 0, 1, \ldots$ processors with $0 \leq \alpha \leq 1$, $\frac{1}{2} \leq \gamma \leq 1$, and $\beta(i,j) \geq 0$ for $i = 0, 1, \ldots, r$ and $j = 0, 1, \ldots, p$ the speedup estimation equals

$$\begin{aligned}&\text{Speedup } R(t,R,P) = \text{Speedup } R(t,2^r,2^p)\\ &\approx \frac{1+\alpha c_X \prod_{i=1}^{r}\beta(i,i)\left(1+\sum_{i=r+1}^{p-1}\prod_{j=r+1}^{i}\beta(r,j)\right)}{2^r+2^{r\gamma}\alpha c_X \prod_{i=1}^{r}\beta(i,i)\left(1+\sum_{i=r+1}^{p-1}\prod_{j=r+1}^{i}\beta(r,j)\right)-2^r\alpha c_X\left(\sum_{i=1}^{r-1}\prod_{k=1}^{i}\beta(k,k)-1\right)},\end{aligned} \tag{5.82}$$

and for the same set of parameters and given that $\beta(i,j) = 1$ for $i = 0, 1, \ldots, r$, $j = 0, 1, \ldots, p$ and $i < j$:

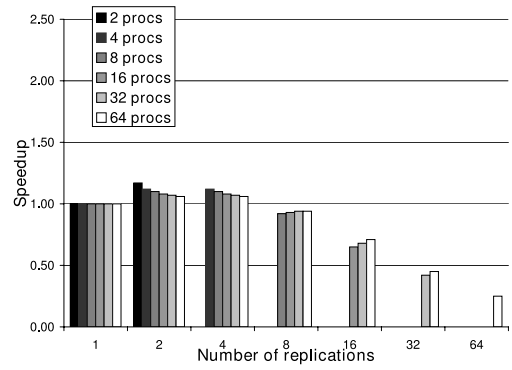
$$\begin{aligned}&\text{Speedup } R(t,R,P) = \text{Speedup } R(t,2^r,2^p)\\ &\approx \frac{1+\alpha p c_X\sqrt{1-2\alpha^2}}{2^r+2^{r\gamma}\alpha c_X\left(\sqrt{1-2\alpha^2}\right)^r(p-r)-2^r\alpha c_X\frac{1-(\sqrt{1-2\alpha^2})^r}{1-\sqrt{1-2\alpha^2}}}.\end{aligned} \tag{5.83}$$

In some cases these speedup formulas derive speedups that are lower than 0 or go to infinity. We expect that for these settings JR leads to a high speedup (> 5). The formula does not provide accurate estimations for those cases. Unfortunately, it is not possible to derive easy equations that illustrate for which parameters the speedup has a value between 0 and infinity.

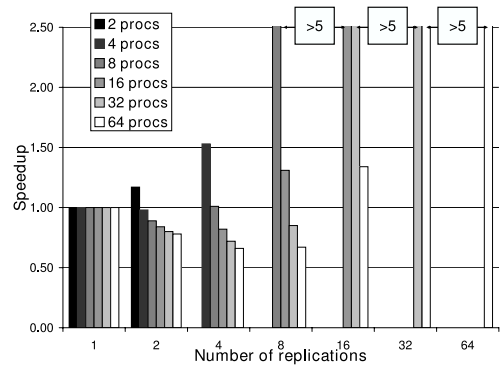
In order to derive an optimal r^* that generally holds we need to take the derivative of (5.82) to r and equate it with 0. Unfortunately, the derivative of this formula does not exist and therefore it is not possible to derive a general formula for the optimal number of replications. Although the derivative of (5.83) to r exists, it is not possible to derive the solution of r^* by equating that derivative with 0.

5.5 Experiments on homogeneous nodes

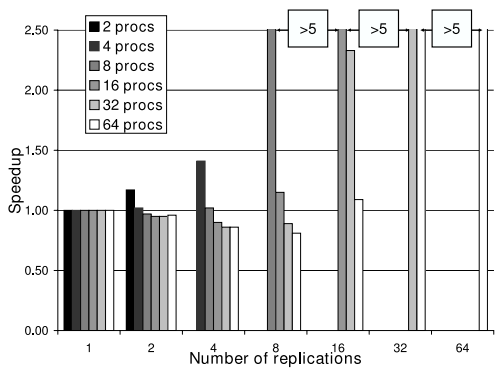
In this section, we perform trace-driven simulations of JR-runs. Furthermore, we use the results of the approximation analyses in Table 5.6 and substitute those in the derived formulas of the previous section which are derived from the assumptions presented in Table 5.7. Moreover, we compare the speedup graphs to those of trace-driven simulations. The simulation details are described in Chapter 2. The data used in this section are the job-runtime measurements as described in Section 2.4. In order to create data as if they are measured on homogeneous nodes, we rescale the data to the same mean and standard deviation. This represents the situation where all nodes have the same capacity, receive on average the same load and where the chances on external factors that influence the job runtimes are the same for each node. In real homogeneous networks, however, the mean and standard deviation of the job runtimes slightly differ per node, which leads to higher speedup than the results show in this section. In order to make a fair comparison between the results of the previous section and of this section, we assume in the implementations of the experiments of this section that send and rescheduling times are negligible. Table 5.8 presents the values of the different parameters that have been applied in the speedup estimations of the different models.



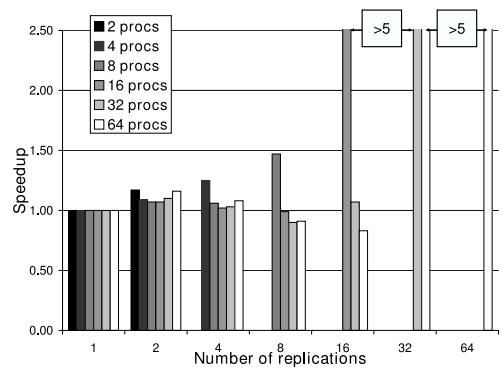
(a) **Model 1.1 with $c_X = 0.80$**



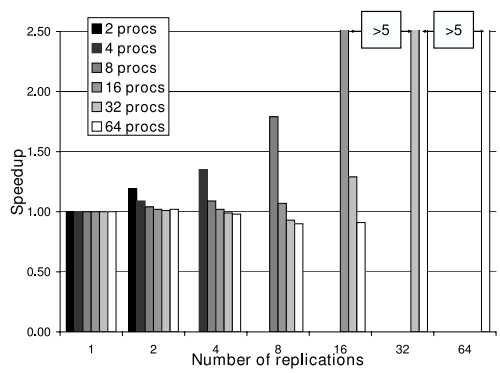
(b) **Model 1.2 with $c_X = 0.80$**



(c) **Model 1.3a with $c_X = 0.80$**



(d) **Model 1.3b with $c_X = 0.80$**



(e) **Simulation with $c_X = 0.80$ data**

Figure 5.16: Speedup estimations of R-JR runs on P processors by models 1.1-1.3b and simulations

Model	Values			
	α	$\beta(i,j),\ \forall i,j : i < j$	$\beta(i,j),\ \forall i,j : i = j$	γ
1.1	$\frac{1}{2}$	1	$\frac{1}{2}$	1
1.2	$\frac{1}{2}$	0.95	0.77, (i.e., $\sqrt{2-2(\frac{1}{2})^2-0.95^2}$)	1
1.3a	$\frac{1}{2}$	$(-0.14i+1.54)(0.02j+0.63)$	$\sqrt{2-2(\frac{1}{2})^2-\beta(i-1,j)^2}$	1
1.3b	$\frac{1}{2}$	Values Table 5.2	$\sqrt{2-2(\frac{1}{2})^2-\beta(i-1,j)^2}$	1
2.1	$\frac{1}{2}$	1	$\frac{1}{2}$	0.81
2.2	$\frac{1}{2}$	0.95	0.77, (i.e., $\sqrt{2-2(\frac{1}{2})^2-0.95^2}$)	0.81
2.3a	$\frac{1}{2}$	$(-0.14i+1.54)(0.02j+0.63)$	$\sqrt{2-2(\frac{1}{2})^2-\beta(i-1,j)^2}$	0.81
2.3b	$\frac{1}{2}$	Values Table 5.2	$\sqrt{2-2(\frac{1}{2})^2-\beta(i-1,j)^2}$	0.81

Table 5.8: Applied values of α, $\beta(i,j)$, and γ for the different models

Figures 5.16(a)-5.16(d) show the results of the speedup estimations by models 1.1, 1.2, 1.3a and 1.3b for a coefficient of variation of 0.80. The histograms depict the speedups for settings with $R = 2^r$, $r = 0, \ldots, p$ replications on $P = 2^p$, $p = 1, \ldots, 6$ processors compared to ELB runs on 2^p processors. Note that because of this definition of the speedups it is possible to make a fair comparison between JR-settings on different numbers of processors. Moreover, Figure 5.16(e) presents the speedups that have been realized by trace-driven simulations. Because of the reason that those results are compared with results of Model 1.1-1.3b, which assume complete dependency between consecutive job runtimes within one iteration, we make the same assumption in the simulations. The exact values of speedup estimations higher than 5 are not presented exactly because we expect that those values are less accurate. Nevertheless, such values indicate that a significant speedup can be gained by JR. The results illustrate that estimations from Model 1.1 are less accurate than those of the other models: this model underestimates the speedups that can be gained by JR. Model 1.2 shows for lower number of replications accurate estimations, and for higher numbers of replications and processors it generates estimations that are higher than the trace-driven simulations. In addition, the estimations of Model 1.3a are for some settings slightly too high, but mostly these are quite accurate. Model 1.3b illustrates very accurate estimations.

Figures 5.17(a)-5.17(d) show the results of the speedup estimations by models 2.1, 2.2, 2.3a and 2.3b for a coefficient of variation of 0.57. The histograms depict the speedups for settings with $R = 2^r$, $r = 0, \ldots, p$ replications on $P = 2^p$, $p = 1, \ldots, 6$ processors compared to ELB runs on 2^p processors. Moreover, Figure 5.17(e) presents the speedups that have been realized by trace-driven simulations. The exact values of speedup estimations higher than 5 are not presented exactly because we expect that those values are less accurate. Nevertheless, such values indicate that a significant speedup can be gained by JR. The results illustrate that estimations from Model 2.1 are less accurate than those of the other models: this model underestimates the speedups that can be gained by JR. Model 2.2 shows for lower number of replications at lower

numbers of processors accurate estimations. However, if the set of processors increases the accuracy of the speedup estimations for the same number of replications decreases slightly. For higher numbers of replications and processors it generates estimations that are higher than the trace-driven simulations. In addition, the estimations of Model 2.3a are for some settings slightly too high, but mostly these are quite accurate. Model 2.3b illustrates very accurate estimations. The results show that extra assumptions for some cases can have a dramatic impact on the accuracy of the speedup estimations, especially for high c_X (above $2/3$).

If the experimental setting contains heterogeneous instead of homogeneous processors, the speedups are higher for the same coefficient of variations. The differences in the processor speeds are higher and therefore JR can make more improvements. For that reason, the speedups presented in this and the previous sections are lower bounds for the speedups on heterogeneous nodes. In the next section, settings with heterogeneous nodes are investigated.

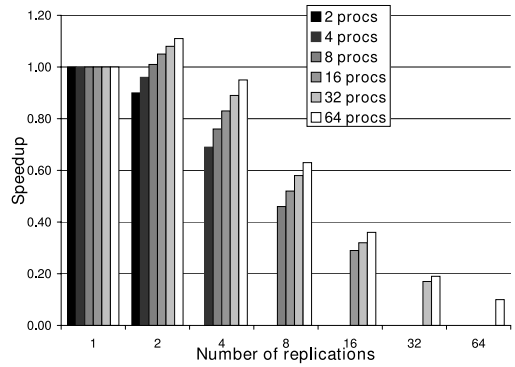
5.6 Experiments on heterogeneous nodes

In the previous sections, we investigated JR in homogeneous settings. In this section, we analyze the performance of JR in heterogeneous settings on the basis of trace-driven simulations. We analyze the expected speedups for different numbers of processors, for different numbers of replications, for the two different sets of datasets, and for the following different CCR values: 0.01, 0.25, and 0.50. Subsequently, we derive the optimal number of replications for the different situations. Table 1 depicts which different situations have been investigated.

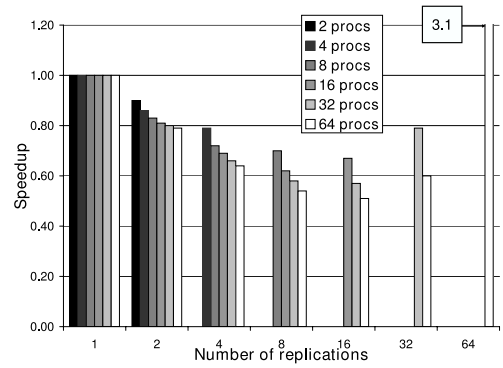
P	Sets	R	CCRs
1	1 2	1	0.01, 0.25, 0.50
2	1 2	1 2	0.01, 0.25, 0.50
4	1 2	1 2 4	0.01, 0.25, 0.50
8	1 2	1 2 4 8	0.01, 0.25, 0.50
16	1 2	1 2 4 8 16	0.01, 0.25, 0.50
32	1 2	1 2 4 8 16 32	0.01, 0.25, 0.50
64	2	1 2 4 8 16 32 64	0.01, 0.25, 0.50

Table 5.9: Used datasets, investigated number of replications and CCRs, for a given number of processors

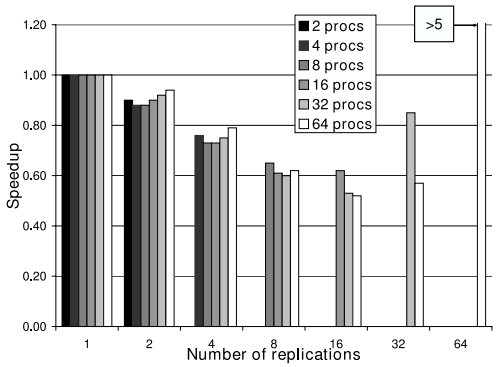
In order to combine and compare previous research performed on DLB in Chapter 4, we use the same sets of data as described in those chapters. set one contains 40 datasets and set 2 the other 90 datasets. set one only consists of nodes in the USA: Boston, Pasadena, Salt Lake City, San Diego, Tucson, and Washington DC. The second set contains, besides the datasets which are generated on the same nodes as set one, datasets which are generated on the following different nodes: Amsterdam, The Netherlands; Cambridge, UK; Beijing, China; Copenhagen, Denmark; Le Chesnay, France; Madrid, Spain; Moscow, Russia; Santa Barbara, USA; Seoul, South Korea; Singapore; Sydney, Australia; Tel Aviv, Israel; Taipei, Taiwan (Academica Sinica);



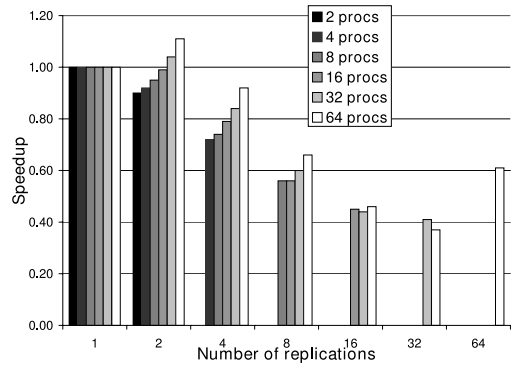
(a) **Model 2.1 with $c_X = 0.57$**



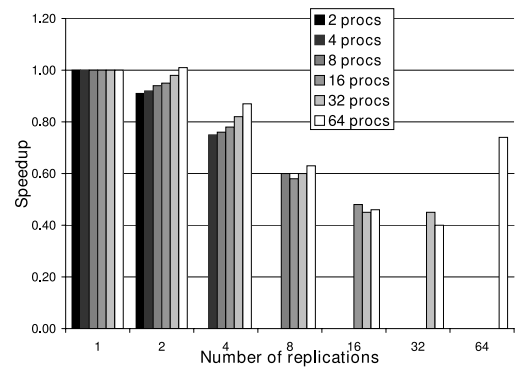
(b) **Model 2.2 with $c_X = 0.57$**



(c) **Model 2.3a with $c_X = 0.57$**



(d) **Model 2.3b with $c_X = 0.57$**

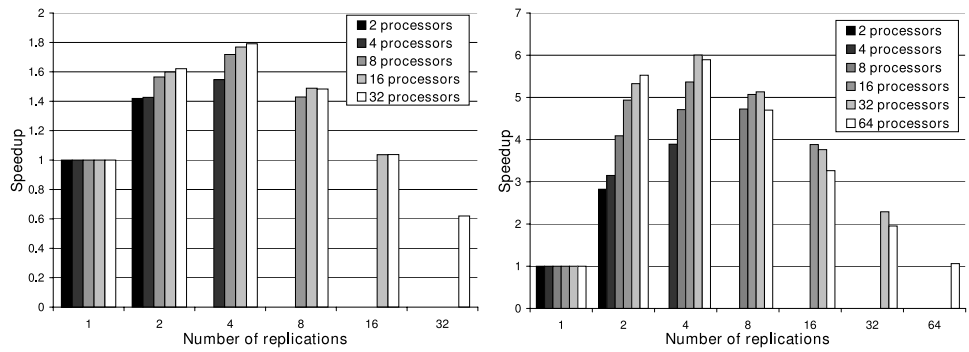


(e) **Simulation with $c_X = 0.57$ data**

Figure 5.17: Speedup estimations of R-JR runs on P processors by models 2.1-2.3b and simulations

Taipei, Taiwan (National Taiwan University); Vancouver, Canada; and Warsaw, Poland. Further analysis shows that the job runtimes on the nodes in set 2 show more burstiness and have higher differences between the average job runtimes on the processors. That last property is mainly caused by the fact that the nodes in set 2 are globally distributed and the nodes in set one are distributed within the USA; set one shows more coherence between the generated datasets. In the trace-driven simulations of this section we include the send and the sending finalize-message times. For further details of the data-collection procedures and the simulation details we refer to Chapter 2.

Next, we show the results of the JR experiments. Figure 5.18(a) and 5.18(b) depict the speedups in the running times of JR on different numbers of processors with CCR 0.01 (the original CCR) for set one and two respectively. We use the definitions of Section 2.5 to compute the speedups, and therefore, it is possible to make a fair comparison between JR-settings on different numbers of processors. For example, the white bars show the speedup that can be gained by JR for different Rs on 32 processors. Subsequently, we present in Table 5.10 the optimal number of replications for different numbers of processors.



(a) **40 datasets of USA nodes** (b) **90 datasets of globally distributed nodes**

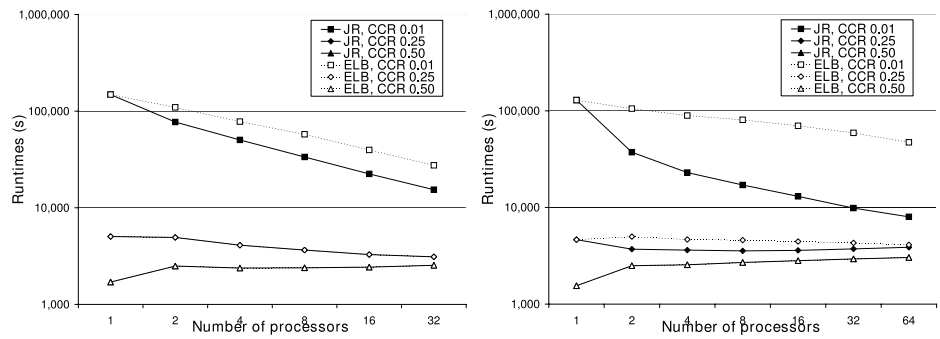
Figure 5.18: Speedups by JR with sets of 40 and 90 datasets

P	R^* (set one)	R^* (set two)
2	2	2
4	4	4
8	4	8
16	4	4
32	4	4
64	-	4

Table 5.10: R^*s for given Ps

We conclude from Figure 5.18(a) and Table 5.10 that for a given number of processors, the speedup increases if 2-JR or 4-JR has been applied. The impact of the fluctuations on the running times is high enough such that 4 replications of each job (i.e., make 3 extra copies of each job) have to be made to maximally decrease the running times. Replicating more than 4 times leads to a speed down compared to a 4-JR run. Furthermore, we conclude that the impact of JR on the speedup for a given number of replications increases if the number of processors has been increased.

The results of the set with 90 datasets, which are shown by Figure 5.18(b) and Table 5.10, show again 4 as the optimal number of replications for most numbers of processors. Except for the runs with 8 processors, as can be seen in Figure 5.18(b), a slightly higher speedup can be gained for the 8- in comparison with the 4-JR case. A difference between the results of this set and the results of the first set is that the speedups are significantly higher. For example, the highest speedup for set one is below the 2.0, while for set 2 even speedups of higher than 6.0 have been registered. This is caused by the differences between set one and two, which is described earlier in this chapter.



(a) **40 datasets of USA nodes** (b) **90 datasets of globally distributed nodes**

Figure 5.19: Running times of JR for different CCR

Furthermore, we simulated the running times of JR on parallel applications with a CCR of 0.25 and 0.50. Figure 5.19(a) and 5.19(b) present the results. We conclude that JR on parallel applications with a CCR of 0.50 never leads to decreases in the running times; the best replication strategy is not to replicate, which equals an ELB run. In addition, JR for the nodes in set one and a CCR of 0.25, JR again does not lead to a running-time decrease. However, the running times on nodes of set 2 and a CCR of 0.25 show in many cases a small decrease of 30% compared to ELB for the best JR strategy. The run times with CCR of 0.50 show a consistent increase in running times if the number of processors increases, which shows that running in parallel in this case is not effective.

5.7 Conclusions

In the present chapter, we modeled SPMD programs that apply job replication (JR) on a set of homogeneous processors. Some very useful and realistic approximations have been introduced and investigated. Extensive computations with the model provide insight in the characteristics and sensitivity of the JR speedups. Comparisons of the results of the computations with the model with those of trace-driven simulations based on real grid testbed measurements show that the model accurately represents JR in real grid environments. Moreover trace-driven simulations of JR in heterogeneous experimental settings illustrate the fundamental speedups that can be gained.

We conclude from the analyses that (1) the speedup that can be gained by JR on heterogeneous nodes is higher than on homogeneous nodes. (2) On a homogeneous set of nodes, if there is no dependency between consecutive jobs on a node, speedup can be gained by JR if the coefficient of variation, c_X, is approximately higher than $2.8/(4.7+0.7p)$, where p is the 2-log of the number of processors. If there is a high correlation between consecutive nodes, speedup can be gained if $c_X > 2/3$ (3) The lower the dependency between consecutive jobs the higher the speedup that can be gained by JR. (4) A small increase in the c_X has a high impact on the speedup. (5) There exists a threshold value c_X^* for which holds that in a resource set with homogeneous nodes that have a c_X higher than this threshold value, the highest speedup can be gained if the number of replications equals the number of processors. For this case the speedup grows at least linearly with the 2-log of the number of processors.

6

AN ADAPTIVE LOAD-DISTRIBUTION STRATEGY

6.1 Introduction

In Chapter 1, we presented the concept of Equal Load Balancing (ELB) which is currently broadly used in parallel applications. In addition, in Chapters 4 and 5, two different implementation types that deal with fluctuations in grid environments have been presented: dynamic load balancing (DLB), and job replication (JR). In this chapter*, we analyze and compare the effectiveness of ELB, DLB and JR, using trace-driven simulations based on real data gathered in a global scale grid testbed, called Planetlab (see Section 2.2). For details of the data-collection procedure and the implementation details of the trace-driven simulations of DLB, JR, we refer to Chapter 2.

This chapter is organized as follows. In Section 6.2, we compare the performance of JR to DLB for different settings. In Section 6.3, we identify a statistic and a corresponding threshold value such that DLB consistently outperforms JR if the statistic is higher than the threshold, whereas JR consistently performs better if it is lower. Furthermore, we develop an adaptive strategy that selects between JR and DLB. Subsequently, we show the results of the experiments with this strategy. Finally, in Section 6.4, we formulate the conclusions.

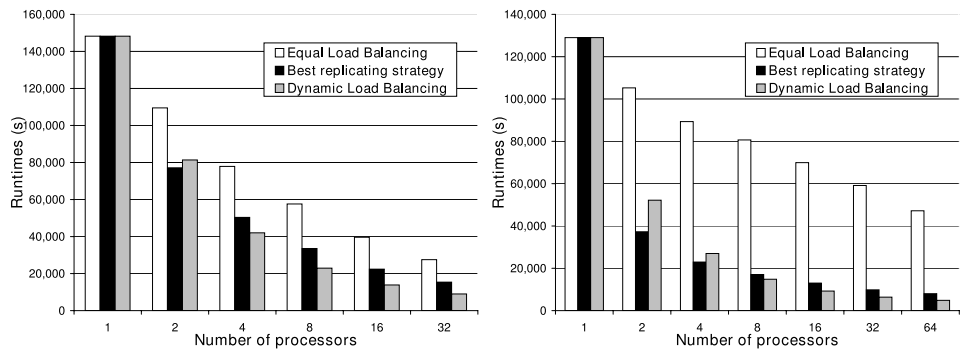
6.2 Comparison between JR and DLB

In this section, we compare the running times of the best JR strategy, the DLB implementation, and of the ELB. In order to combine and compare previous research performed in Chapters 4 and 5, we use the same sets of datasets as described in those chapters. Set one contains 40 datasets, which are generated from nodes in the USA, and set two contains 90 datasets, which are gathered on nodes located in all over the world. For more details about the nodes we refer to Sections 4.3 and 5.6. Analysis shows that the job runtimes on the nodes in set two show more burstiness and have higher differences between the average job runtimes on the processors. That last prop-

*This chapter is based on paper [27].

erty is mainly caused by the fact that the nodes in set two are globally distributed and the nodes in set one are distributed within the USA; set one shows more coherence between the generated datasets. For further details of the data-collection procedure we refer to Chapter 2.

Next, we analyze and compare the effectiveness of ELB, DLB and JR, using trace-driven simulations based on real data gathered in Planetlab. For implementation details of the trace-driven simulations of DLB, JR, we refer to Chapter 2. Figures 6.1(a) and 6.1(b) depict the running times of the above mentioned three different strategies for processors set one and set two respectively.

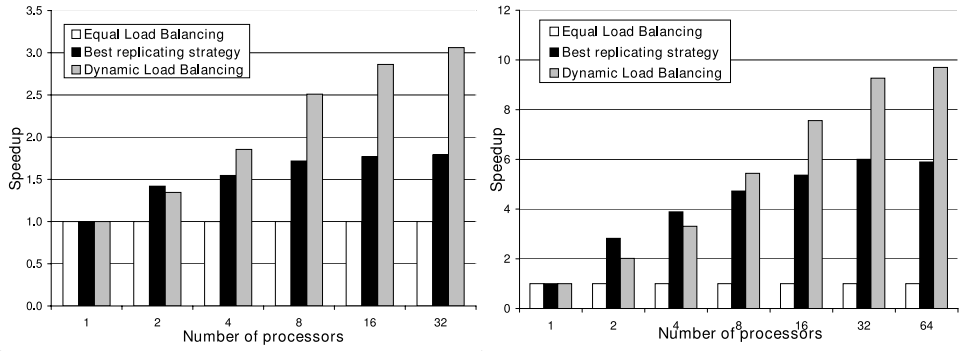


(a) **40 datasets of USA nodes** (b) **90 datasets of globally distributed nodes**

Figure 6.1: Run times of DLB and JR with CCR 0.01

From Figures 6.1(a) and 6.1(b) can be concluded that the running times of JR and of DLB are consistently lower than those of the ELB implementation. We conclude that if the CCR equals 0.01, deploying more processors for the same amount of load leads to a speedup for all three types of implementations and for both sets of datasets. Figure 6.1(b) depicts super-linear speedups for JR and DLB if two processors are used instead of one. This is caused by the effect that if one processor is used, peaks in the job runtimes have a dramatic impact on the total runtime of the application. In runs with two processors this effect can be reduced by the faster second processor. A difference between the results of set one and set two is that the running times of the DLB and JR implementations of set two decrease faster while the running times of ELB decrease slower. This is caused by the differences between set one and two, which is described above.

For further analysis, we compute the speedups, as defined in Chapter 2, of the best JR strategy and DLB for set one and two from the running times from Figures 6.1(a) and 6.1(b). Figures 6.2(a) and 6.2(b) depict those computed speedups. Figure 6.2(a) shows for the simulations with set one that DLB consistently outperforms or at least performs as good as JR. We conclude from Figure 6.2(b) that in comparison with the results of nodes from set one significantly higher speedups can be gained on the nodes of set two. This insight corresponds to the observations mentioned above in this section. Moreover, we notice for the experiments with the nodes of set two that the best JR strategy has a higher speedup than DLB for runs on 2 or 4 processors.



(a) **40 datasets of USA nodes** (b) **90 datasets of globally distributed nodes**

Figure 6.2: Speedups of DLB and JR with sets of 40 and 90 datasets with CCR 0.01

However, when more processors are used, DLB outperforms all JR strategies. This is mainly caused by the fact that when more processors are used, the amount of load per processor decreases and, as a consequence, the load that has to be redistributed during the DLB rescheduling phase decreases. For this reason, the overhead time of DLB shows more sensitivity to the number of processors. Those numbers are the results of the following trade-off: on the one hand, when more processors have been deployed, the load per processor and, therefore, the gain by DLB, decreases. On the other hand, the probability on slow processors increases, which delay the whole process in ELB implementations. We observe that the results of set one show that the speedup of DLB has its maximum at 16 processors and for set two the maximal speedup can be gained when 32 processors are used. This is again caused by the differences in average processing times between the nodes and the higher fluctuations over time. We remark that the results of Figure 6.2(a) are consistent with the results in [26]: the DLB runs on four randomly selected processors of set one show again on average a speedup of 1.8. Comparing the results of DLB and JR for the different CCR 0.01, 0.25, and 0.50 show that for a CCR of 0.01, there are some circumstances for which replication shows the best results. However, for CCRs of 0.25 and 0.50 DLB clearly outperforms JR for all situations. For many of the cases, replication does not even show speedups in comparison with ELB. We conclude that when the CCR increases, the gain that can be obtained by JR insufficient to the compensate the overhead and extra computations of this method. On the other hand, the overhead of DLB remains low enough when the CCR increases, to be able to gain speedups.

6.3 The adaptive load-distribution strategy

Next, we introduce the concept of an adaptive strategy that selects between the above two different types of implementations. This strategy has the aim to select dynamically the optimal implementation type. The idea behind the strategy is that the performance of DLB and JR is strongly related to the expected computation time. To this end, it measures a statistic Y, which is the expected computation time per iteration, and

defines a threshold Y^* during the run, which is the intersection of the performance of DLB and the JR against the Y. After each I_p iterations, the strategy gives a preference for a given type of implementation, based on a comparison of Y with the threshold value Y^*. In this strategy, the processor that redistributes the load in DLB is moreover the processor that decides whether JR or DLB is used. When the strategy decides that a switch to the other type of implementation is necessary, the steps to be taken are the same as in the DLB rescheduling phase: (1) the nodes send their prediction to the scheduler, (2) the scheduler computes the optimal load distribution, and (3) the nodes redistribute their load.

6.3.1 Dynamic adaptive load-distribution strategy

In this section, we first analyze the opportunity to develop a adaptive load-distribution strategy that is based on a threshold value. Second we propose a adaptive load-distribution strategy that optimally selects between the two implementation types: DLB and JR.

6.3.1.1 Analysis

To be able to develop such a adaptive load-distribution strategy, we need to find formulas that indicate the height of the iteration times for the different implementations for a given easy-to-measure statistic. To this end, we first derive an approximation of the expected iteration time for DLB, which is based on the predictions $\hat{y}_t$, by adding the expected ST and rescheduling time to (2.26).

$$D_{it}(N, f, t, P) \approx \frac{1}{\frac{1}{P}\sum_{i=1}^{P} \frac{1}{\mathbb{E}\hat{y}_i}} + \mathbb{E}ST + \frac{\mathbb{E}RSchT}{N}. \tag{6.1}$$

However, this expectation differs from the real measured iteration times, mainly due to inevitable differences between the expected job runtimes and the realized job runtimes. Therefore, we define an equation for the expected iteration time of DLB ($D_{it}(N, f, t, P)$) with a- and b-values which take into account those differences. We assume that the send- and rescheduling times can also be included in the b-values.

$$D_{it}(N, f, t, P) = a_{D_{it}(N,f,t,P)} \frac{1}{\frac{1}{P}\sum_{i=1}^{P} \frac{1}{\mathbb{E}\hat{y}_i}} + b_{D_{it}(N,f,t,P)}. \tag{6.2}$$

Furthermore, because of the reason that the iteration time of JR also has a strong linear relation to (2.26), we define the expected iteration time of JR as:

$$R_{it}(t, R, P) = a_{R_{it}(t,R,P)} \frac{1}{\frac{1}{P}\sum_{i=1}^{P} \frac{1}{\mathbb{E}\hat{y}_i}} + b_{R_{it}(t,R,P)}. \tag{6.3}$$

We address that the a- and the b-values generally depend on (1) the MSE between the predicted and the realized job runtimes, (2) the distribution of the send times, (3) the distribution of the rescheduling times, and (4) the values of the parameters R and P.

Next, we investigate the relation between the average of 100 realized iteration times provided by realistic trace-driven simulations and the estimations of the iteration times

by the above equations. An effective statistical property to quantify this dependency is the correlation coefficient. This property can be derived by substituting the data values of both quantities in the correlation formula. The correlation coefficient varies by definition between the -1.0, which indicates a complete negative linear dependency, and the 1.0, which indicates a complete positive linear dependency. A correlation of 0.0 indicates no linear dependency. More details can be found in [45]. The high correlation of 0.97 between those values implies that equations (6.2) to (6.3) are accurate indications of the possibly realizable speedups. Consequently, this means that for a given implementation and a given choice of parameters (e.g., number of processors), the speedup strongly depends on the statistic, which is defined in (2.26). In the remainder of this chapter, we call this statistic Y ($= P/\sum_{i=1}^{P} \frac{1}{\mathbb{E}JT_i}$).

We consider the following situation. We perform 1000 simulations of a 4-JR implementation on 4 nodes and 1000 simulations of a DLB implementation on 4 nodes with 10 as the number of iterations between two load rescheduling phases. For comparison reasons, the 1000 simulations of the DLB implementation have been executed on the same set of nodes as on the corresponding JR simulation. 1000 as the number of simulations is high enough in order to derive stable and reproducible results. Figure 6.3 depicts the averages of 100 iteration times of those trace-driven simulated runs of DLB and JR against statistic Y.

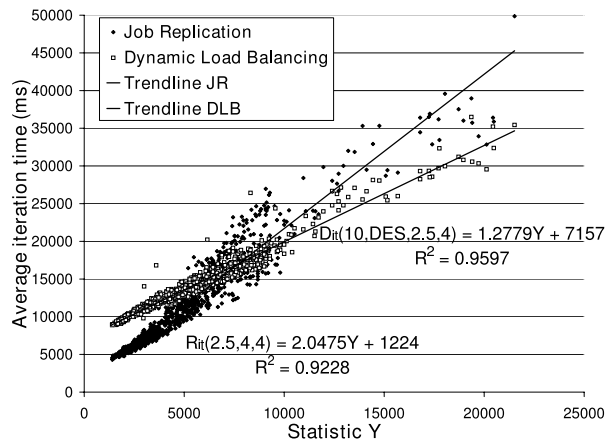


Figure 6.3: Scatter plot of DLB and JR iteration times

Figure 6.3 shows, as expected, the strong linear relation between the statistic Y and the iteration times. We fitted trend lines by a least-squares fit and derived the R^2 values of those equations, which will both be explained at the end of this section. The trend line of the iteration times of JR has the following equation:

$$R_{it}(2.5, 4, 4) = 2.0475Y + 1224, \tag{6.4}$$

with a R^2-value of 0.9228, and for DLB the trend line equals

$$D_{it}(10, DES, 2.5, 4) = 1.2779Y + 7157, \tag{6.5}$$

with a R^2-value of 0.9597.

The above observations imply the possibility of deriving a threshold value Y^* of statistic Y that defines the optimal implementation choice by equating both equations. The threshold policy works as follows: when statistic Y is lower than this threshold, we choose for JR and when Y is higher, we choose for DLB. The threshold for the above situation would be: $Y = 7708$, which is the solution of equating the formulas for $R_{it}(2.5, 4, 4)$ with $D_{it}(10, DES, 2.5, 4)$.

The strategy

Given the above equations, we are able to develop a strategy that dynamically chooses the most effective implementation from both DLB or JR. We propose the following adaptive load-distribution strategy:

Step 1: Start with DLB as the current choice.

Step 2: Measure for the current implementation choice the job- and iteration times during I_p iterations.

Step 3: Estimate the job- and iteration times for the other as the current implementation by straightforward computations, which are shown in Chapter 2.

Step 4: Compute statistic $Y = P/\sum_{i=1}^{P} \frac{1}{\mathbb{E}JT_i}$.

Step 5: Fit the values of $a_{D_{it}(N,f,t,P)}$, $a_{R_{it}(t,R,P)}$, $b_{D_{it}(N,f,t,P)}$, and $b_{R_{it}(t,R,P)}$ in equations (6.2)-(6.3) by a least-squares fit (more information below) of the collected data about the iteration times and the statistics [75]. When only one data point has been collected, go to step eight and take the iteration times of the first I_p iterations as expected iteration times for the next I_p iterations.

Step 6: In order to derive an expectation of the iteration times of the different implementations, take the latest computed value of Y and substitute it in (6.2)-(6.3) with the fitted values of $a_{D_{it}(N,f,t,P)}$, $a_{R_{it}(t,R,P)}$, $b_{D_{it}(N,f,t,P)}$, and $b_{R_{it}(t,R,P)}$.

Step 7: Choose the implementation with the lowest expected iteration time for the next I_p iterations.

Step 8: If the run is not finished, go to step two.

A least-squares fit is an effective method that fits a linear equation to a collection of data points. It is based on minimizing the sum of the squares of the deviations between the linear equation and the data points. The values of a and b can be derived by a direct formula of the values of the data points. The R^2 value is an indication of the overall deviation between the trend line and the data points, and ranges between 0.0 (no fit) and 1.0 (complete fit). We refer to [45] for the formulas for a, b, and the R^2.

We chose to take $I_p = 100$ as the number of iterations between two implementation-evaluation steps. On the one hand, this number is low enough to react fast on a change in the best implementation type. On the other hand, an implementation with this number involves relatively low overhead costs that is caused by the

switch procedure between DLB and JR.

The results of extensive analysis of iteration-time predictions have shown that an estimation which is based on substituting measurements of Y in (6.2)-(6.3) is far more accurate than taking the average iteration time of the last 100 iterations.

6.3.2 Adaptive strategy experiments

In this section, we present the results of the trace-driven simulations of the adaptive strategy implementation. This strategy selects dynamically during the run between the two implementations DLB and JR. The details of this strategy are described in 6.3.1. For example, we perform an experiment with DLB, and 4-JR on 4 processors. Figure 6.4 shows us for this experiment the derived values of statistic Y for the following different groups of iteration numbers: $1-100, 101-200, \ldots, 1901-2000$. Moreover, the figure depicts the corresponding realized iteration times of DLB and JR.

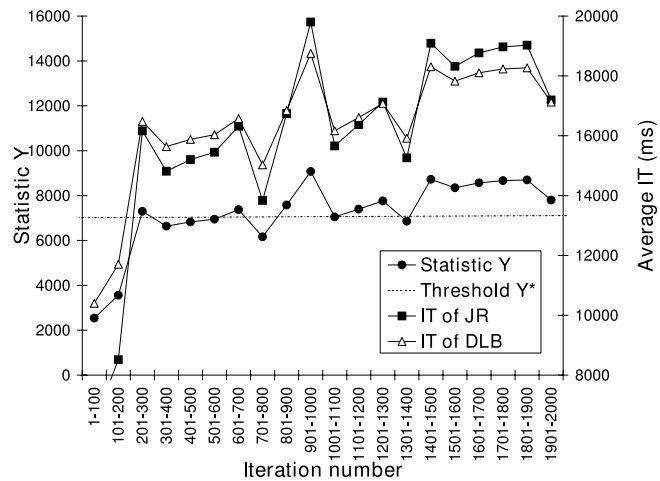


Figure 6.4: Statistic Y against iteration times of DLB and JR

As we have seen above, the threshold value Y^* for the comparison between a JR-run with 4 replications and a DLB run on 4 processors is 7708. Figure 6.4 shows that for this situation, the Y is lower of equal than 7708 until iteration number 900. This means that JR has the lowest running times, which corresponds to the measured average iteration times for these iteration numbers. Until iteration number 1400, the statistic Y moves around the threshold value and therefore both implementations can be used. Likewise, the realized iteration times of both implementations do not differ significantly. After iteration number 1400, the threshold value clearly moves above the threshold which indicates that DLB is the best choice, because of lower iteration times.

Finally, we compare the speedups of the adaptive load-distribution strategy with those of the DLB and JR implementations. To this end, we performed 1000 experiments with the adaptive load-distribution strategy on 1, 2, 4, 8, 16, 32, and 64 randomly chosen nodes from set two. Figure 6.5 depicts the speedups of the strategy compared to those of DLB and JR.

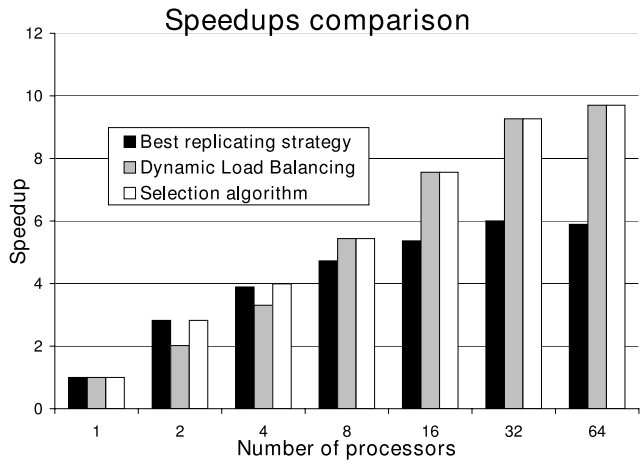


Figure 6.5: Speedup of adaptive load-distribution strategy, DLB and JR

We conclude that the strategy that selects between DLB and JR performs at least as good as both DLB and JR for all situations. The overhead of the switches to the best performing method is in every experimental setting completely compensated by the gain in running time resulting from those effective switches. For the cases in which one of the two implementation types is significantly faster, the performance of the adaptive load-distribution strategy exactly equals the highest possible performance, because for those situations it immediately selects the one with the highest speedups. Summarizing, the results in Figure 6.5 show that the introduced dynamic strategy is very effective in making Bulk Synchronous Processing parallel programs robust against the fluctuations of a globally distributed grid environment and in which there is no knowledge about which of the methods JR or DLB will perform as best.

6.4 Conclusions

In summary, in this chapter we first made a comparison between DLB and JR. The results show that both DLB and JR strongly outperform the default ELB, which is widely deployed in grid environments today. Further, a comparison between DLB and JR reveals that in some circumstances JR performs better than DLB, but in other circumstances DLB is preferable. Given the strong unpredictability of the circumstances in the grid environment, this observation makes it difficult to assess the relative effectiveness of DLB or JR. Nonetheless, we found that there exists an easy-to-measure statistic Y and a corresponding threshold value Y^* such that DLB outperforms JR for $Y > Y^*$, whereas JR consistently performs better for $Y < Y^*$. Based on this observation, we proposed a simple and easy-to-implement approach that can make on-the-fly decisions about whether to use to DLB or JR. Simulations based on a large set of real data in a global-scale grid show that this new dynamic approach always performs at least as good as both DLB and JR in all circumstances. As such, the new approach presented provides a promising means to make parallel applications robust in large-scale grid environments.

7

PREDICTION METHODS

7.1 Introduction

In this chapter*, we focus on the development of a new method to predict the running times of jobs on shared processors. In Chapter 3, we investigated the statistical properties of the job runtimes generated on globally distributed grid nodes. In Section 7.2, on the basis of both this statistical analysis and theoretical analysis we investigate a broad set of predictors (e.g., NWS, AR-methods, Whybark, STES predictors), and identify the strengths and weaknesses of those predictors. The analysis reveals that the main sources of inaccuracy of these existing methods are (1) over reaction to sudden peaks in the running times, and (2) delayed reaction to "level switches". Further, in Section 7.3, based on these observations we develop a new prediction method, called Dynamic Exponential Smoothing (DES). The main idea behind DES is that it uses ES where the interpolating factor α has a number of levels that can be dynamically adapted to the height of the outliers in the data. Subsequently, in Section 7.4 we compare the accuracy of the predictions resulting from DES to that of the other prediction methods. For the comparison we use 45 datasets of 18 different Planetlab nodes. The results show that DES strongly outperforms the existing methods in the vast majority of the datasets. Furthermore, in Section 7.5, the relation between the quality of the DES predictor and the statistical properties of the datasets is investigated. Finally, in Section 7.6 we make some concluding remarks.

7.2 Analysis of existing prediction methods

In this section, we analyze a variety of existing methods that can be used to predict the running times of jobs on shared processors. In Chapter 3, one of the main conclusions was that the datasets of job runtimes contain peaks, which correspond to a sudden very high or very low job runtimes, and level switches, which correspond to 4 or more successive running times of jobs that significantly differ from the previous values. Accurate predictors of job runtimes omit peaks and adjust the prediction fastly if level switches occur. Taking this into account and the other conclusions about the

*This chapter is based on papers [28], and [29].

statistical properties of job runtimes, we identify the shortcomings of the existing prediction methods. To this end, we use the following six datasets: two of runs that ran on a node at Sydney (au01 and au02), one of a run in Tel Aviv (telaviv01), two at Tucson (ar01 and ar02), and one at Warsaw (warsch01). The details of how the datasets are generated are described in Section 2.4. In this section, the same abbreviations are used as that are introduced in that section.

The most commonly used predictors in grid studies are ES [26, 64], NWS [71, 72, 73] and AR [23, 61] (these predictors will be described below). Nevertheless, for predicting the running times of tasks there are many other useful predictors applicable from other areas (e.g., economics). We selected the Adaptive Exponential Smoothing Predictors (AESP) and Smooth Transition Exponential Smoothing (STES) predictors as potential methods that can give accurate predictions in grids. From the AESP we discuss Trigg and Leach [67], Whybark [70], Mentzer [55], Pantazopoulos and Pappis [58], and from the STES predictors we discuss STES $|e|$ with $\gamma < 0$, STES $|e|$, STES e^2, and STES Whybark, which we selected from [66]. Below, we describe the different prediction methods and assess their strong and weak points for making forecasts based on the characteristics of the datasets discussed in the previous section.

7.2.1 Common grid predictors

In this section, we consider the most commonly used predictors in grid environments: Exponential Smoothing (ES), Network Weather Service (NWS) and Autoregression (AR).

7.2.1.1 Exponential Smoothing

ES is a simple prediction method that surprisingly often works very well in practice. We define y_t as the measured value at time t, $\hat{y}_t$ as the prediction for y_t, α as a chosen parameter between 0 and 1. Next, the prediction for y_t is defined as:

$$\hat{y}_t = \alpha y_{t-1} + (1-\alpha)\hat{y}_{t-1}, \text{ where } 0 \leq \alpha \leq 1. \tag{7.1}$$

An advantage of ES is that it contains an α value which can be low if the dataset contains a lot of peaks, such that it does not heavily adapts the prediction to sudden peaks, and which can be high if there are many level switches in the runtimes, such that it adjusts the forecast fastly if these switches occur. This means that the most suitable value of the interpolation parameter α depends on the characteristics of the dataset. A weak point is that once a parameter is chosen, it always reacts in the same way to peaks and level switches, even when the structure is changed completely, which were found to occur frequently in Section 2. For example, Figure 7.1 shows that ES with parameter 0.5 does not react properly to a peak, even when there were many peaks in the history data. In this chapter we use the ES parameter 0.5, because in [25] is stated that 0.5 is the value that leads to the most accurate predictions for running times of jobs.

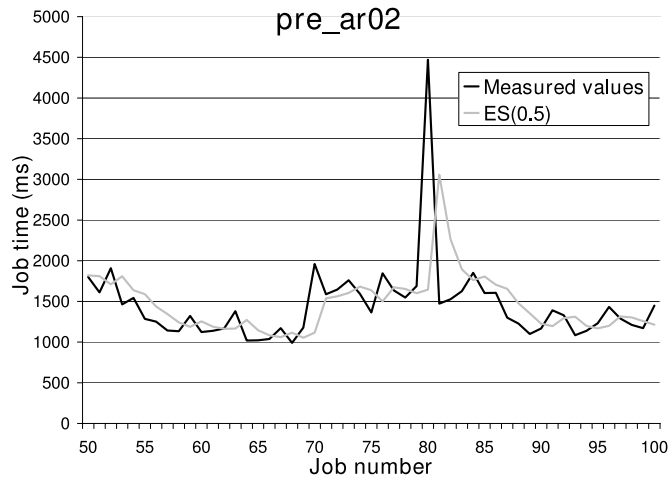
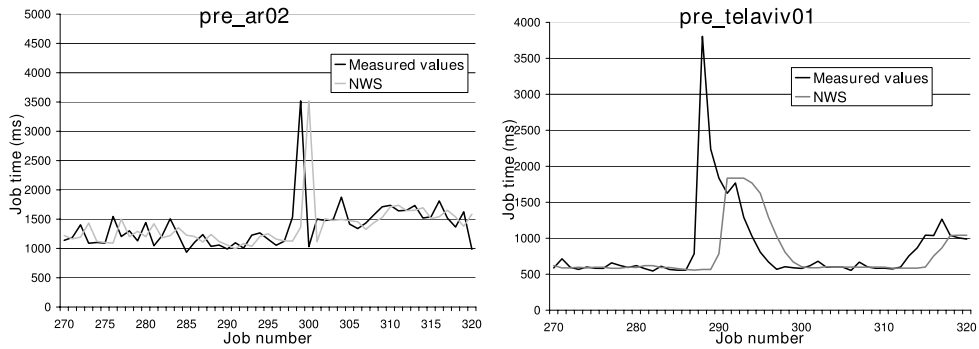


Figure 7.1: ES reacting on a peak on pre_ar02

7.2.1.2 The Network Weather Service

The Network Weather Service (NWS) prediction method conducts post casts using different windows of previous data (always starting with the most recent data and working backwards in time) and records the ”winning” forecaster for each window size. A post cast is a forecast of a property at a historical point in time. The prediction accuracy can be immediately measured. Each window with historic data is subsequently treated as a separate forecaster and a final accuracy tournament determines which forecaster will be used. The predictor selects from the following set of predictors: the adaptive median window 5-21 and 21-51, 30% trimmed median window 31 and 51, sliding median window 5 and 31, median window 5 and 31, the running mean, the last value, exponential smoothing with trend 10% and parameter 30%, 20%, 15%, 10%, and exponential smoothing with parameter 90%, 75%, 50%, 40%, 30%, 20%, 15%, 10%, 5%. Further details can be found in [72].

A strong point of this method is that it contains a large set of predictors. They represent many different characteristics of the datasets. Therefore, the NWS prediction method is able to deliver accurate predictions in many situations. Further investigations with the 6 analysis-phase datasets show that some parts of the datasets contain more than 20% peaks, or show an alternating characteristic. For those cases a predictor based on the average is the most accurate. Some parts of the datasets show a homeostatic character, for which the sliding median with a window size of 31 gives the best predictions. Both types (i.e., the average and the sliding median with a window size of 31) of predictors are represented in the NWS set. Another good point of this selection method is the case of changing characteristics, this predictor rapidly chooses another predictor that predicts the new situation more accurately. A weak point is that it does not always react properly to peaks. Figure 7.2(a) shows that when there were small level switches in history, the NWS method often chooses a predictor that over reacts to peaks. That is even the case when peaks in history never introduced big changes.



(a) **Peak on pre_ar02** (b) **Level switch on pre_telaviv01**

Figure 7.2: NWS-predictor reacting on a peak and a level switch in the running times

Moreover, Figure 7.2(b) shows that the NWS prediction method reacts too slowly on the new level, despite many peaks introduced a new level for previous values. Clearly, peaks in datasets have to be interpreted differently than regular values. However, the NWS does not make a distinction between those types of values.

7.2.1.3 Autoregression

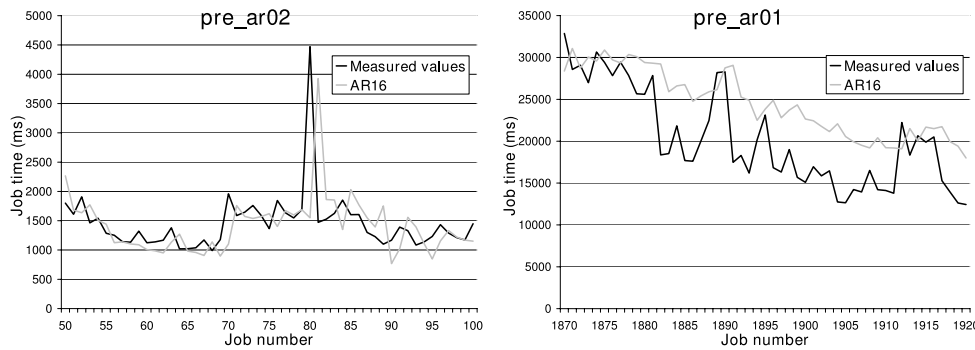
General Autoregressive (AR) predictors multiply previous data-points with some parameter between 0 and 1 to compute the next prediction. AR predictors have a parameter p which indicates the number of data-points it uses from the history. Generally, AR predictions (denoted as $AR(p)$) are calculated in the following way:

$$\hat{y}_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \ldots + \alpha_p y_{t-p}, \text{ with } -1 \leq \alpha_i \leq 1. \tag{7.2}$$

The α parameters are carefully chosen, based on historical correlations between y_t's over different time-scales. When the expectation of the y_t's does not equal 0, the mean of the y_t's is subtracted from those y_t's and finally added to the $\hat{y}_t$. The sum of the α_i values does not necessarily equal 1.0. Dinda [23] has also analyzed a set of other prediction models, related to AR models, and showed that $AR(16)$ performs the best in forecasting processor loads and running times of tasks. However, as stated in [23], they only work well in case of periodicity. As we have seen in Figures 3.5(c) to 3.5(b), we notice that the datasets do not show periodicity. Dinda used k-steps-ahead prediction methods and implemented the Yule-Walker technique. However, we only use the part of Dinda's software that implements the one-step ahead prediction methods for the following two reasons. The first reason is the ability to compare the predictions with the one-step-ahead predictions of other predictors and the second reason is that one-step ahead predictions are significantly more accurate.

A strong point is that AR methods adapt the parameters to the characteristics. In case of many peaks, this method will adapt the parameters in a more 'averaging' way, such that the peaks do not influence the predictions too much. When more level switches occur, this method chooses for higher values for the first α parameters, such

that it reacts fast on those switches. A weak point, however, is that the choice of the parameters is not optimal. For example, when the AR method takes a high first parameter because the set shows some small level switches, it will give a highly inaccurate prediction when a higher peak occurs (see Figure 7.3(a)). Another drawback of this method is that when the structure of data points is changed it takes a long time before the method is adapted due to the fact that this method uses more than 100 data points to fit the parameters. Also in case of a trend upwards or downwards (see for example Figure 7.3(b)) this method adapts the predictions too slowly, because a significant part of the prediction is based on the average of the whole dataset.



(a) **Peak on pre_ar02** (b) **Trend down on pre_ar01**

Figure 7.3: AR16-predictor reacting on a peak and a trend down in the running times

7.2.2 Adaptive Exponential Smoothing Predictors

Adaptive Exponential Smoothing Predictors (AESP) [66] use the following formula:

$$\hat{y}_t = \alpha_{t-1} y_{t-1} + (1 - \alpha_{t-1})\hat{y}_{t-1}. \tag{7.3}$$

and focus on adapting α_t such that the α_t will always get a good value that is independent of the start value, and adapts when the structure of the values changes. We consider the following variants of AESPs: Trigg and Leach [67], Whybark [70], Mentzer [55], Pantazopoulos and Pappis [58], and the STES predictors [66].

7.2.2.1 Trigg and Leach

The method of Trigg and Leach [67] defines the smoothing parameter as the absolute value of the ratio of the smoothed forecast error to the smoothed absolute error.

$$\hat{y}_t = \alpha_{t-1} y_{t-1} + (1 - \alpha_{t-1})\hat{y}_{t-1}, \tag{7.4}$$

with

$$\begin{aligned} \alpha_t &:= |\frac{A_t}{M_t}|, \\ A_t &:= \phi(y_t - \hat{y}_t) + (1 - \phi)A_{t-1}, \\ M_t &:= \phi|y_t - \hat{y}_t| + (1 - \phi)M_{t-1}, \end{aligned}$$

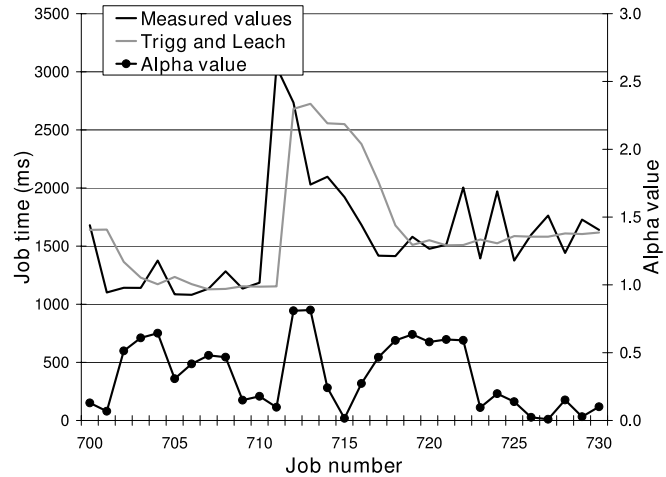


Figure 7.4: A trend up and down on pre_ar02

where ϕ is set arbitrarily, with 0.2 being a common choice [67]. Trigg and Leach explain that this formulation enables α_t to vary according to the degree to which biased forecasts are obtained.

A main advantage of this prediction method is that it chooses a low α parameter when there is a lot of noise in the dataset and a high α when there is less noise and there are some level-switches. These properties improve the accuracy of the predictor. A disadvantage of the proposed method is that it is difficult to find a suitable choice of the parameter ϕ and that the approach sometimes delivers unstable forecasts [32]. Another weak point is that when a few successive data-points show a trend up (in Figure 7.4 from value 710 to 711) and then a trend down (see Figure 7.4 from value 711 to 717) in the α value will first increase to close to 1, which increases the accuracy, and then decreases to 0 and finally increases back to 1. Consequently, as is shown by the Trigg and Leach prediction values 713 to 718 in Figure 7.4, the predictor does not give very accurate predictions. In this case, it would be better when the α-value was kept fixed to 1. Similar problems occur when there are many level switches up- and downwards.

7.2.2.2 Whybark

AESP Whybark [70] defines the control limits in terms of multiples of the forecast error standard deviation, σ. An indicator variable, δ_t, is defined as:

$$\delta_t := \begin{cases} 1 & \text{if } |y_t - \hat{y}_t| > 4\sigma, \\ 1 & \text{if } |y_t - \hat{y}_t| > 1.2\sigma,\ |y_{t-1} - \hat{y}_{t-1}| > 1.2\sigma, \text{ and } (y_t - \hat{y}_t)(y_{t-1} - \hat{y}_{t-1}) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

The value of δ_t determines whether α_t takes a base value, B, a medium value, M, or a high value, H. Whybark suggests $B = 0.2, M = 0.4$ and $H = 0.8$. Taking

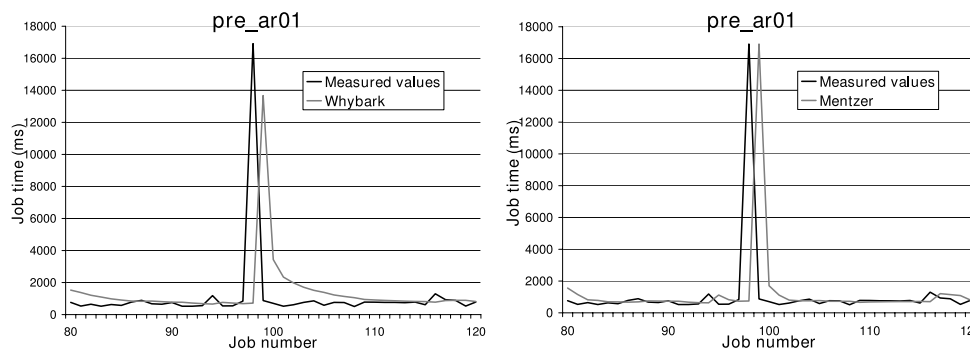
everything together, the Whybark prediction is computed in the following way:

$$\hat{y}_t = \alpha_{t-1} y_{t-1} + (1 - \alpha_{t-1})\hat{y}_{t-1}, \tag{7.5}$$

with

$$\alpha_t := \begin{cases} H & \text{if } \delta_t = 1, \\ M & \text{if } \delta_t = 0, \text{ and } \delta_{t-1} = 1, \\ B & \text{otherwise.} \end{cases}$$

A strong point of Whybark is that it distinguishes between normal situations and those where there was a huge difference between the prediction and the measured value; characteristics of predictions are always different when big differences occur. However, further analysis with the 6 datasets shows that the Whybark predictor gets a higher accuracy when it would distinguish between (1) 'normal' fluctuations, (2) fluctuations that are higher than 2 times the measured standard deviation, and (3) fluctuations that are higher than ten times the measured standard deviation. A second strong point is that Whybarks' predictor contains a *correction part*; that is, when two successive measurements both deviate in the same direction with more than 1.2 times the standard deviation from the predictions, a different α_t value has to be applied. Further analysis with our analysis-phase datasets shows that this correction part increases the accuracy of the predictor, but that a value of 1.0 makes better distinctions. A weak point is that it does not distinguish between up- and downward fluctuations; it is not likely that those situations can be interpreted the same. Moreover, Whybark reacts in the same way to huge peaks as to two successive small differences between the prediction and the measured value. Furthermore, Whybark always uses the same parameters for the same kind of peaks; 0.8 for high peaks, 0.4 for the value after a high peak and also for two successive differences, and 0.2 as a base value. In practice, however, characteristics for each dataset are different. Figure 7.5(a) shows that the Whybark predictor does not react properly to peaks. Whybark will only work well when peaks introduce a new 'level' of values. It would be smarter when Whybarks predictor learns from historical data whether a huge difference introduces a new level or is a peak.



(a) **Whybark reacting on the peak** (b) **Mentzer reacting on the peak**

Figure 7.5: Two predictors reacting on a peak in pre_au01

Finally, we conclude that the Whybark predictor contains a useful fluctuations classification, but that the following improvements can be made: (1) other decision rules to distinct between the 3 classes of fluctuations, (2) a correction part parameter of 1.0, (3) different treatment of down- and up-wards fluctuations, (4) different treatment of a data point after a huge peak and two successive significant, but not enormous, differences, and (5) adaptive parameters.

7.2.2.3 Mentzer

AESP Mentzer [55] uses the absolute forecast error fraction from the most recent period as α_t. In order to restrict α_t to the interval [0, 1], if the absolute error fraction exceeds 1.0, then α_t is set to 1:

$$\hat{y}_t := \alpha_{t-1} y_{t-1} + (1 - \alpha_{t-1})\hat{y}_{t-1}, \text{ with } \alpha_t := \min\left(\left|\frac{y_t - \hat{y}_t}{y_t}\right|, 1\right). \tag{7.6}$$

A strong point of this method is that when the forecasting error increases (decreases), the α value also increases (decreases) to improve the forecast. A weak point of this method is the reaction on peaks: when a peak occurs, the predictor will have a large forecasting error, which leads to a high next prediction, which in turn causes another large prediction error. This is illustrated in Figure 7.5(b). Another weak point is that it is not clear why there should be a 1- 1 relation between the α-value and the last forecasting error. For example, a forecasting error of 50% is a worse prediction and probably needs an α value that is higher than 0.5. The height of the α value also depends on the height of the standard deviation of the value.

7.2.2.4 Pantazopoulos and Pappis

Pantazopoulos and Pappis [58] argue that, since the ideal value for α_t would lead to $\hat{y}_{t+1} = y_{t+1}$, this ideal value can be derived by substituting y_{t+1} for $\hat{y}_{t+1}$ in (7.3) and solving for α_t to give

$$\alpha_t := \left(\frac{y_{t+1} - \hat{y}_t}{y_t - \hat{y}_t}\right). \tag{7.7}$$

Since y_{t+1} is unknown at time t, the ideal value of α_t for period $t-1$ is used for period t to give

$$\alpha_t := \left(\frac{y_t - \hat{y}_{t-1}}{y_{t-1} - \hat{y}_{t-1}}\right), \tag{7.8}$$

which can be substituted into the standard ES-Formula (7.3). In order to restrict α_t to the interval [0, 1], Pantazopoulos and Pappis propose that if $\alpha_t \notin (0, 1)$ then α_t is set to either 0 or 1, whichever one is closer.

A strong point of this predictor is that it calculates the optimal value of the α_t's for the last predictor-value combination. With high probability, that optimal choice of α_t is also a very good choice for the next value. Unfortunately, brief consideration of (7.8) suggests that the approach is unlikely to be of use. Since the one-step-ahead forecast is also the multi-step-ahead forecast for simple exponential smoothing, the numerator

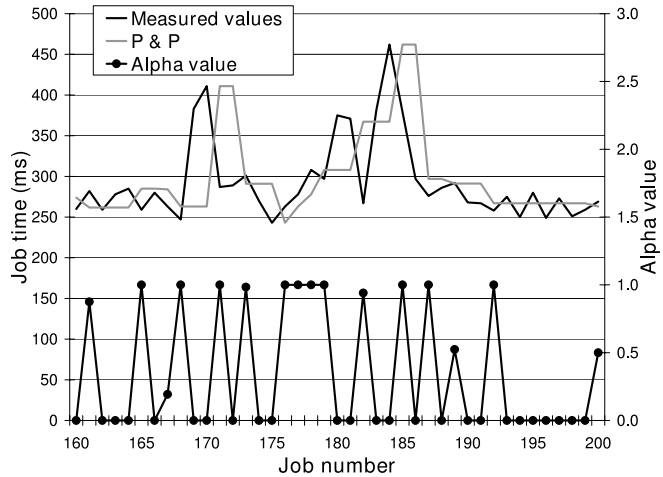


Figure 7.6: P&P-predictor in pre_warsch01

is a two-step-ahead forecast error, while the denominator is a one-step-ahead forecast error. This suggests that the expression in (7.8) will very often lead to values larger than 1. Following the rule of Pantazopoulos and Pappis (P&P), α_t would then take a value of 1. The result is that α_t very often takes a value of 1, which is shown by Figure 7.6. Moreover, this figure shows that the interpolation parameter α of the P&P predictor is quite unstable, and in many cases results in inaccurate predictions, see for example data points 171, 172, 185 and 186.

7.2.3 STES predictors

A smooth transition exponential smoothing (STES) method [66] has a smoothing parameter α_t defined as a logistic function of a user-specified transition variable, V_t. Mathematically, a STES method is defined as:

$$\hat{y}_t = \alpha_{t-1} y_{t-1} + (1 - \alpha_{t-1})\hat{y}_{t-1}, \tag{7.9}$$

where

$$\alpha_t := \frac{1}{1 + e^{\beta + \gamma V_t}}. \tag{7.10}$$

If $\gamma < 0$, then α_t is a monotonically increasing function of V_t. Hence, as V_t increases, the weight on y_t increases, and consequently, the weight on $\hat{y}_t$ decreases. The logistic function restricts α_t to lie between in the interval (0,1). Historical data is used to calibrate the adaptive smoothing parameter, α_t, through the estimation of β and γ in (7.10). The derived values for β and γ in (7.10), govern the degree to which the variation in the transition variable influences the STES smoothing parameter. The choice of the transition variable, V_t, is of crucial importance to the success of the method. Consideration of the adaptive methods described in the previous section leads to a number of different possible transition variables. The value of the smoothing parameter in all of the existing adaptive methods depends to varying degrees on the magnitude of the

most recent periods forecast error. At the end of this section we describe the different methods we used: STES $|e|$, STES e^2, and STES Whybark.

In [66], 80 data points are used to fit the parameters, and those parameters are used to make one-step-ahead predictions of the next 20 values. To make a fair comparison between the STES- and the other predictors, we fitted the β and the γ after every new measured value. Moreover, for the STES predictors, we compared 10, 80, and "all values" as the number of data points used for the fit.

Next, we describe the three specific STES prediction methods we used in the predictor analysis.

STES $|\mathbf{e}|$

An obvious choice for the transition variable is the absolute value of the forecast error from the most recent period:

$$V_t := |y_t - \hat{y}_t|. \tag{7.11}$$

In our analyses in Section 7.4 we make distinction between the situation in which we let the predictor fit β and γ freely, and in which we restrict the γ to be lower than 0. The purpose of restricting the γ to negative values is to get a high α value when the absolute error is high.

STES $\mathbf{e^2}$

Another obvious choice for the transition variable is the square of the forecast error from the most recent period:

$$V_t := (y_t - \hat{y}_t)^2. \tag{7.12}$$

STES Whybark

It is possible to use the α_t outcome values of all the AESPs. We choose to use the Whybark parameter as the transition variable, because short analysis showed that the Whybark parameter was the best transition variable to choose:

$$V_t := \begin{cases} H & \text{if } \delta_t = 1, \\ M & \text{if } \delta_t = 0 \text{ and } \delta_{t-1} = 1, \\ B & \text{otherwise,} \end{cases} \tag{7.13}$$

with

$$\delta_t := \begin{cases} 1 & \text{if } |y_t - \hat{y}_t| > 4\sigma, \\ 1 & \text{if } |y_t - \hat{y}_t| > 1.2\sigma,\ |y_{t-1} - \hat{y}_{t-1}| > 1.2\sigma, \text{ and } (y_t - \hat{y}_t)(y_{t-1} - \hat{y}_{t-1}) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

A strong point of a STES predictors discussed above is that it does not assume a linear relation between the transition variable and the value α_t. With the logistic function and the freedom of the parameters β and γ many different relations can be created. Another strong point is that the relation between the transition variable and the α_t's is fitted with two parameters, based on the history. Therefore, the α_t is adapted

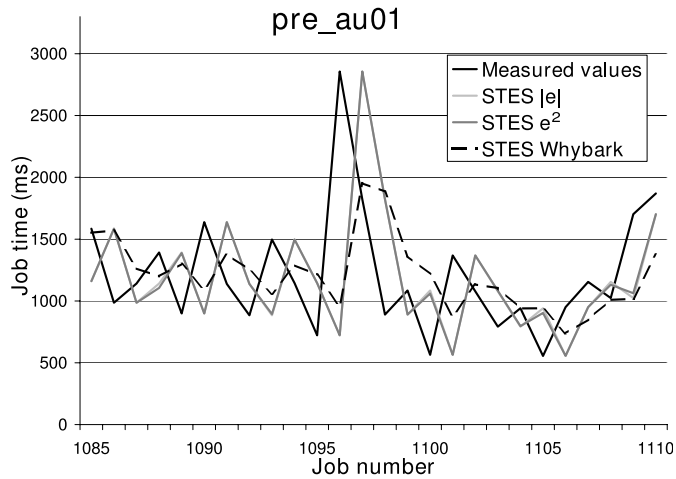


Figure 7.7: STES predictors reacting on a peak in pre_au01

in a way that it is optimal according to the history. Moreover, when the transition variable has no correlation with the right α_t, the method automatically adapts the γ to 0 and the smoothing parameter will be constant. The STES method, therefore, enables re-calibration of the existing adaptive methods.

A weak point, however, is that it is not clear why a logistic function would work better than other functions. Although previous experimental results have shown that logistic functions work quite well, it is the question whether it is an optimal function for the values of running times of jobs. Further on, this method uses the same formula for peaks as for stable situations. Figure 7.7 shows the predictions around a peak of three different STES predictors, which are described below. The figure illustrates that using the same parameters for peaks as for stable situations leads to bad predictions. Besides, the peaks have a high impact on the optimal parameters β and γ. Consequently, the α will behave in a stable situation highly influenced by previous peaks.

7.2.4 Other predictors

As stated in [74] a homeostatic or a tendency-based predictor can be very effective in predicting CPU load. A homeostatic predictor uses the assumption that if the current value is greater (less) than the mean of the history values, then the next value is likely to decrease (increase). A tendency-based predictor uses the assumption that when a current value is greater (less) than the mean of the history values, then the next value is likely to increase (decrease), because of a trend up- or downwards.

A disadvantage is that before the prediction process is started, a choice between homeostatic or tendency has to be made. We know from the data analysis that it is possible that the characteristics in the dataset change, and that during the run the other type of predictor seemed to be more accurate. During our data analysis we noticed that trends do not appear very often in the datasets, there are many level switches, and sometimes homeostatic situations appear. For that reason we think that tendency-based

predictors are not very accurate in predicting the running times of jobs. A homeostatic predictor will probably work fine for some datasets, but for the nodes with many level switches this method is very unlikely to be effective in the grid context. For these reasons, we have not implemented this prediction method.

Another well-known method is Linear Regression (LR). This method is quite similar to AR, when only historical data is used: the predictor is computed by multiplying historical data with parameters, which are fitted by using a least-squares method and the history. When not only data from the previous history is used, but also other information, like load, this method differs from AR. The parameter used to multiply with other data will also be fitted by its historical data. Since we focus on prediction methods based on the use historical data of running times of jobs only, LR is beyond the scope of this study.

Besides the linear prediction-methods, some experiments on a non-linear prediction method, called a Neural Network, have been performed. The results of the experiments illustrate that Neural Networks are computationally infeasible and are not accurate enough to be useful for prediction purposes in grids.

7.2.5 Conclusions of analyses

We conclude that although each prediction method has its strong points, none of the predictors properly reacts to peaks and level switches, which are two of the most important characteristics of the evolution of running times in a grid environment. This raises the need for the development of a new prediction method particularly suited for the specifics of a grid environment. For this reason, in the next section we use the insights in the pros and cons of the existing prediction methods to develop a new prediction method that effectively reacts to the peaks and level switches observed in a real grid environment.

7.3 New prediction method

7.3.1 DES prediction method

In this section we propose a method that effectively reacts to the peaks and level switches: the Dynamic Exponential Smoothing (DES) method. To this end, we take into account the benefits and drawbacks of the existing prediction methods. The basic idea of the DES prediction method is that it (1) treats different classes of fluctuations differently, (2) always computes the optimal parameter for those classes, and (3) has the possibility to select another predictor from a set, because of performance reasons. The parameter details are described below the mathematical description of the method. Furthermore, the method to derive the optimal historical α parameter, α_t^*, is described in Section 7.3.2. Finally, in Section 7.3.3 we discuss the effectiveness of the proposed predictor. We propose the following complete DES prediction method for $t = 2, 3, \ldots$:

$$\hat{y}_t = \begin{cases} \hat{y}_{DES,t} & \text{if } \kappa_{DES} \quad = min_{j \epsilon \{DES,\mu,median\}}(\kappa_j), \\ \hat{y}_{\mu,t} & \text{if } \kappa_{\mu} \quad = min_{j \epsilon \{DES,\mu,median\}}(\kappa_j) \neq \kappa_{DES}, \\ \hat{y}_{median,t} & \text{otherwise,} \end{cases} \tag{7.14}$$

with

$$\hat{y}_{DES,t} := \alpha^*_{t-1} y_{DES,t-1} + (1-\alpha^*_{t-1})\hat{y}_{DES,t-1}, \tag{7.15}$$

$$\hat{y}_{\mu,t} := \mu((y_1,\ldots,y_{t-1})), \tag{7.16}$$

$$\hat{y}_{median,t} := median((y_{t-l},\ldots,y_{t-1})), \text{ for } l < t \tag{7.17}$$

$$\kappa_i := \sum_{s=1}^{t-1}(y_s - \hat{y}_{i,s})^2, \;\; i \,\epsilon\, \{DES, \mu, median\}, \tag{7.18}$$

where

$$\alpha^*_t := \frac{1}{\sum_{s=2}^{t}(y_{s-1}-\hat{y}_{s-1})^2 1_{\{\delta_s=\delta_t\}}} \sum_{s=2}^{t} \alpha_{s-1,opt}(y_{s-1}-\hat{y}_{s-1})^2 1_{\{\delta_s=\delta_t\}}, \tag{7.19}$$

where

$$\alpha_{s-1,opt} := \frac{y_s - \hat{y}_{s-1}}{y_{s-1} - \hat{y}_{s-1}} \tag{7.20}$$

$$\delta_s := \begin{cases} H1 & \text{if} & |y_s - \hat{y}_s| & > 10\sigma((y_{s-k},\ldots,y_{s-1})), \\ H2 & \text{if} & (y_s - \hat{y}_s) & > 2\sigma((y_{s-k},\ldots,y_{s-1})), \\ H3 & \text{if} & (y_s - \hat{y}_s) & < -2\sigma((y_{s-k},\ldots,y_{s-1})), \\ M & \text{if} & |y_s - \hat{y}_s| & > \sigma((y_{s-k},\ldots,y_{s-1})), \\ & & |y_{s-1} - \hat{y}_{s-1}| & > \sigma((y_{s-k},\ldots,y_{s-1})), \\ & \text{and} & (y_s - \hat{y}_s)(y_{s-1} - \hat{y}_{s-1}) & > 0, \\ B & \text{otherwise}, & & \end{cases} \tag{7.21}$$

Note that $\mu((y_1,\ldots,y_{t-1}))$ is defined in (3.4), $\sigma((y_{t-k},\ldots,y_{t-1}))$ is defined in (3.5), and $median((y_{t-l},\ldots,y_{t-1}))$ is defined in Definition 2.5.6.

As can be seen in (7.14), the DES prediction method selects the best prediction method from a set of 3 predictors, by comparing the sum of the squared errors of the previous measurements. The set contains the running average, the l-sliding window median, and the DES predictor. The l-sliding window median takes the median of the last l measurements. The implemented value for l is presented below. The DES-predictor part classifies different types of fluctuations: three huge fluctuations classes ($H1$, $H2$, and $H3$), 1 medium fluctuations class (M), and 1 base class (B). If the deviation between the measured value and its prediction is more than ten times the standard deviation of the last k values, the measurement is of class $H1$. Furthermore, when the deviation between the measurement and the prediction is higher than two times (smaller than minus two times) the standard deviation, the measurement is of class $H2$ ($H3$). Two consecutive measurements that both deviate in the same direction with more than standard deviation from the predictions are of class M. B is the base class and is defined for the rest of the cases. This classification is defined in (7.21). When a measurement fits into multiple classes, the first mentioned class in (7.21) will be chosen. Subsequently, the DES predictor computes the prediction by using the exponential-smoothing equation with an α parameter that equals the weighted average of the optimal α's (see (7.19)) of the previous measurements that are of the same class as the most recent measurement. Formula (7.19) is derived below.

We implemented the predictor with $k = 20$, and $l = 31$, and fixed the maximal number of data points to estimate each of the optimal α values for classes $H1$, $H2$, $H3$, M and B at 500. When no fluctuations of a certain type were registered before, the default value of 0.5 is taken. On the one hand, when k is higher than 20, the predictor reacts slower on changes in standard deviations. On the other hand, when less values are taken, the estimated standard deviations get less accurate. While 20 is a good value for k, a higher or lower value does not affect the results significantly. The value 31 as the value for l corresponds to one of the set of medians in [71]. Accuracy comparisons showed that this sliding-window median performs better than medians with other l parameters. The same holds for the choice to use all previous measurements for the computation of the average. The number 500 ensures that on the one hand the α_t^* is reliable and stable, because it is computed by enough measurements, and that on the other hand the α_t^* is still able to adapt to new characteristics in the dataset.

7.3.2 Computation of α_t^*

In this section, we derive (7.19). To this end, let $\hat{y}_{s,opt}$ be the 'best possible prediction' of y_s, for $s \in \{2, \ldots, t\}$. Then by definition,

$$\hat{y}_{s,opt} = y_s. \tag{7.22}$$

Moreover, let $\alpha_{s-1,opt}$ be the best possible parameter for prediction of y_s. Then

$$\hat{y}_{s,opt} = \alpha_{s-1,opt} y_{s-1} + (1 - \alpha_{s-1,opt})\hat{y}_{s-1}. \tag{7.23}$$

We compute the optimal exponential smoothing parameter for the prediction of y_s, $\alpha_{s-1,opt}$, by solving (7.23) and replacing y_s by $\hat{y}_{s,opt}$, according to (7.22). We derive the following $\alpha_{s-1,opt}$.

$$\alpha_{s-1,opt} = \begin{cases} \frac{y_s - \hat{y}_{s-1}}{y_{s-1} - \hat{y}_{s-1}} & \text{if } y_{s-1} - \hat{y}_{s-1} \neq 0, \\ 0 & \text{else.} \end{cases} \tag{7.24}$$

At a given time step s, if y_s is known, we are able to calculate the Squared Error (SE_s) of one prediction in the following way:

$$\begin{aligned} SE_s &= (y_s - \hat{y}_s)^2 = (\hat{y}_{s,opt} - \hat{y}_s)^2 \\ &= (\alpha_{s-1,opt} y_{s-1} + (1 - \alpha_{s-1,opt})\hat{y}_{s-1} - (\alpha_{s-1} y_{s-1} + (1 - \alpha_{s-1})\hat{y}_{s-1}))^2 \\ &= (\alpha_{s-1,opt} - \alpha_{s-1})^2 (y_{s-1} - \hat{y}_{s-1})^2. \end{aligned} \tag{7.25}$$

These formulas hold for all $s \in \{1, \ldots, t\}$, with t the current time step. Subsequently, we derive the total MSE of all predictions y_s, for all $s \in \{1, \ldots, t\}$. To this end, we assume that $\alpha_s = \alpha$ for all $s \in \{2, \ldots, t\}$.

$$MSE_t = \frac{1}{t-1} \sum_{s=2}^{t} (\alpha_{s-1,opt} - \alpha)^2 (y_{s-1} - \hat{y}_{s-1})^2. \tag{7.26}$$

Next, we calculate α_t^*, i.e., the value of α in (7.26) that minimizes the MSE_t, by taking the setting derivative of (7.26) to α to 0. This leads to the following expression for α_t^*:

$$\alpha_t^* = \frac{1}{\sum_{s=2}^{t} (y_{s-1} - \hat{y}_{s-1})^2} \sum_{s=2}^{t} \alpha_{s-1,opt} (y_{s-1} - \hat{y}_{s-1})^2. \tag{7.27}$$

Note that this is equivalent to the weighted average of the values of $\alpha_{s-1,opt}$, with weights $(y_{s-1}-\hat{y}_{s-1})^2$, for $s = 2, \ldots, t$. Finally, to derive (7.19), an indicator $1_{(\delta_s=\delta_t)}$ is added to (7.27) to distinguish between the different classes.

7.3.3 Discussion

Formula (7.14) shows the selection algorithm of the DES prediction method. As we have seen for the NWS method in Section 7.2.1.2, the datasets sometimes show characteristics for which the average or the sliding median with window size 31 gives the most accurate predictions. Around 10 % of the measurements show these kinds of characteristics. Analysis shows that measuring the κ_i's (see Formula (7.18)) is an effective way of comparing the different predictors: in all the 10 % of the cases the average or the sliding-window median is chosen. We note that comparing the κ_i's is the same as comparing the Root of the Mean Squared Errors (as will be described in Section 7.4).

In Chapter 3, we observed that the datasets may have completely different and continuously changing characteristics. Therefore, as can be seen in Formula (7.19)), we chose to adapt the α_t's to the situation of the characteristics. We concluded from the analysis of the AESPs, the STES predictors and the AR method in Section 7.2 that adapting the α_t to the characteristics is an effective way to make the predictor robust against all the completely different and ever-changing characteristics of the datasets. However, we showed that improvements were possible on the following aspects: the choice of the α_t was not always clear and the α_t's are unstable in many predictors. The choice of taking the optimal α of the measured data is very clear and leads to stable α_t's within the classes of fluctuations.

Next, as shown in (7.21) we classified different types of fluctuations. We concluded in Section 7.2.2.2 that Whybark consists of useful classification elements, but that improvements are necessary to be made to develop a useful classification of running times. We incorporated those improvements in our prediction method, as can be seen in (7.21).

Moreover, easy to measure statistics can be included in the model that compute the MSE and the RMSE of the DES predictor by computing the squared differences between the predictions from (7.14) and the data measurements. These statistics can be applied in an online scheduling system that dynamically decides on the basis of expectations of job runtimes which nodes in the resource set are used.

Finally, we would like to address that this method can be extended for more general situations. We have planned to make those analyses in further research, which is described in Chapter 8.

7.4 Experimental results

In this section we compare the performance of the DES prediction method with those of the predictors described in Section 7.2. The following 45 datasets are used for those accuracy comparisons: Amsterdam (ams), Beijing (china), Le Chesnay (inria), Copenhagen (dk), Madrid (mad), Moscow (mos), Pasadena (cal), Salt Lake City (utah), San Diego (sandiego), Santa Barbara (santab), Singapore (sing), Sydney (au), Taipei (tw), Tel Aviv (telaviv), Tucson (ar), Vancouver (ca), Warsaw (warsch), and Washington

(wash). The details of how the datasets are generated are described in Section 2.4. In this chapter, the same abbreviations are used as that are introduced in that section.

7.4.1 Quality metrics for prediction methods

To make a fair comparison we first have to select a metric for the prediction error. There are two commonly used error evaluators: the Root of the Mean Squared Error (RMSE) and the Mean Absolute Errors (MAE). Sometimes the MSE is taken instead of the RMSE. We selected the RMSE because in the kind of applications we used it is very important to minimize the number of high forecasting errors. Since the RMSE is more sensitive to high forecasting errors, we prefer to use this metric. The RMSE of a given predictor Y is defined as

$$RMSE_Y = \sqrt{\frac{1}{N-1}\sum_{t=2}^{N}(y_t - \hat{y}_t)^2} \tag{7.28}$$

We also need to define a metric that quantifies the improvement of a predictor in comparison to another predictor. As we have seen in Figure 3.8(e), jumps have a strong impact on the standard deviation or the RMSE of the dataset, which is highly correlated with the standard deviation. When the peaks or level switches do not show up periodically, it is hard to predict when they occur. Consequently, jumps always have a strong effect on the RMSE of a predictor. For that reason, we are interested in a measure that is able to compare the performance of different predictors and is independent of the influence of the jumps on the standard deviation. Therefore, to compare different predictors we define $\Delta\%(A, B)$ to be the percentage improvement of predictor A versus B:

$$\Delta\%(A, B) := \frac{RMSE_B - RMSE_A}{RMSE_B - RMSE^*} * 100\%, \tag{7.29}$$

where the value $RMSE^*$ is the optimal *post-cast* selected from the set of the NWS predictors. It indicates the theoretically maximal forecasting performance (minimum error) that the method could have achieved if the best predictor at each step was known (see for more details about the $RMSE^*$ the definition of the *Optimum* in [71]). When a jump occurs, even the optimal forecasting method has the property that it was not able to predict that jump. But for the successive measurement, the optimal forecast mostly has a low RMSE. Consequently, the $RMSE^*$ is the RMSE that a prediction method would have when it would predict the behavior after jumps perfectly.

7.4.2 Comparison results

To compare the performance of DES with the existing prediction methods, we have gathered experimental data from 45 datasets, as discussed above in this section. We first compare the predictors on the basis of the time that it takes to compute each prediction. Second we extensively compare their accuracies. The results are outlined below.

Table 7.1 shows us the ranking for the different prediction methods in terms of the time that it takes to compute the prediction for the next value, and it shows the number of computations needed per prediction. For the NWS and the AR predictor

those times have been measured in [23] and [72], which is shown by the last column. The table illustrates that the DES needs to perform a lower number of computations than the NWS, which takes 161 microseconds on a Pentium III laptop. Consequently, we conclude that the computational costs of the DES predictor are low enough such that it can feasibly be implemented in online systems.

	Predictor	Number of computations per prediction	Computational time
1	ES	two multiplications and one summation	$<$ 161 microseconds
2	AESP	nr. of multiplications and summations	$<$ 161 microseconds
3	DES	nr. of multiplications and summations (order of 10)	$<$ 161 microseconds
4	NWS	nr. of multiplications and summations (order of 100)	[72]: 161 microseconds on an unloaded 750 MHz Pentium III laptop.
5	AR	fitting procedure	[23]: 1.4 ms on an unloaded 500 MHz Alpha 21164-based workstation
6	STES	fitting procedure	$>$ 1.4 ms

Table 7.1: Comparison of the computational costs of the prediction methods

Next, we compare the accuracy of the different prediction methods. Figure 7.8 shows the performance improvements of the DES prediction method compared to the Trigg and Leach predictor and the Whybark predictor. Similarly, Figure 7.9 shows the results for DES compared to the Mentzer and Pantazopoulos & Pappis predictor.

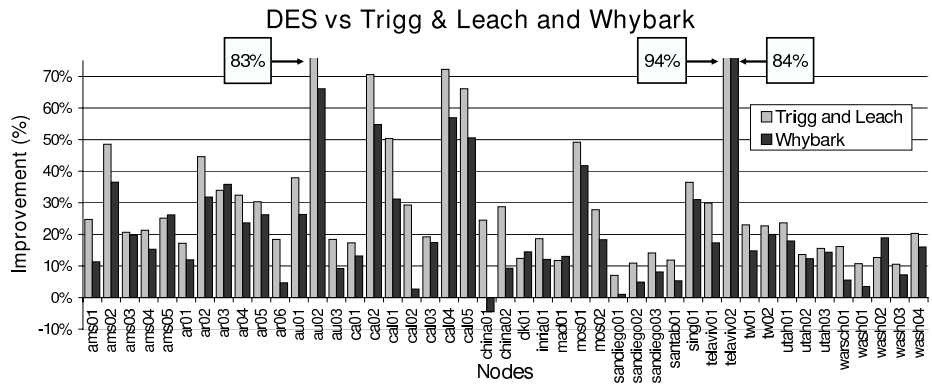


Figure 7.8: DES-predictor improvements compared to the Trigg & Leach and the Whybark predictor

The results presented in Figures 7.8 and 7.9 show that the DES prediction method strongly and consistently outperforms the other predictors. The improvements are remarkably high: for many datasets the DES prediction method shows improvements of

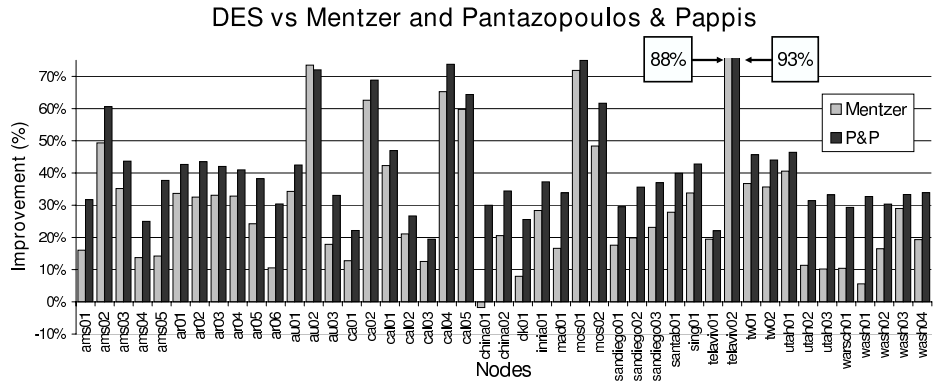


Figure 7.9: DES-predictor improvements compared to the Mentzer and Pantazopoulos & Pappis predictor

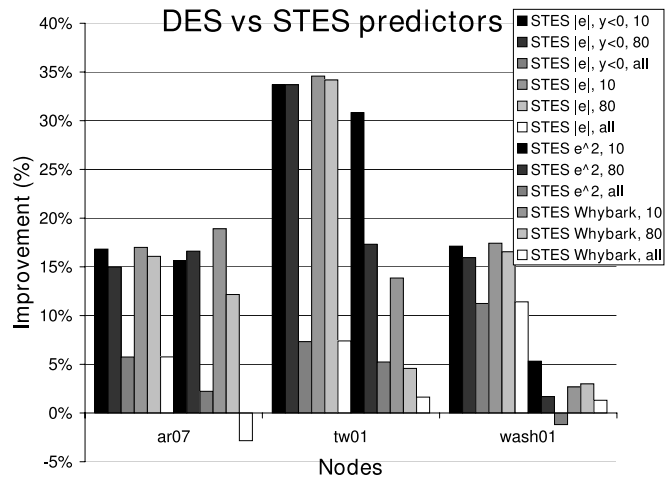


Figure 7.10: DES-predictor improvements compared to the STES predictors

more than 30%. From the four AESP predictors considered here, the Whybark predictor gives the most accurate predictions, but is still strongly outperformed by our DES predictor.

Figure 7.10 shows the prediction performance of in total 12 STES predictor-parameter combinations for the three randomly-chosen datasets ar07, tw01, and wash01. We tested the STES |e| predictor with parameter $\gamma < 0$, the STES |e| predictor with no restrictions on γ, the STES e^2 predictor and the STES Whybark predictor. For all the four predictors we used 10, 80 and 2000 data points for the fit. We tested those predictors only with three datasets due to the fact that the STES predictors take too much time (a whole day per dataset). We observe considerable differences in the prediction results of the different methods. Although three datasets are not enough to draw conclusions that are statistically significant, we expect because of those significant differences that the following observations hold for almost all the datasets. The

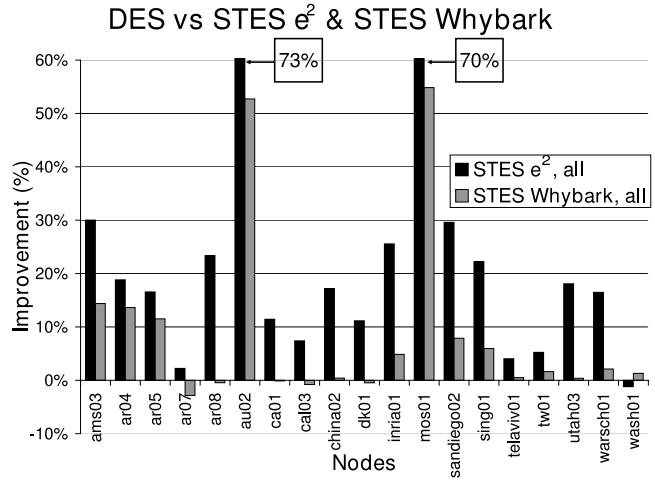


Figure 7.11: DES-predictor improvements compared to the STES e^2 and STES Whybark predictors

results in Figure 7.10 show for the three datasets that it is better to use all the data (at least 2000 values) for the parameter fit, and that the STES e^2 and the STES Whybark are the best STES predictors. Those methods need further comparison analysis with more datasets to conclude more about the quality of their predictions. We used the following randomly-chosen datasets for the further analysis: ams03, ar04, ar05, ar07, ar08, au02, ca01, cal03, china02, dk01, inria01, mos01, sandiego02, sing01, telaviv01, tw01, utah03, warsch01, and wash01. In Figure 7.11 we compare the improvements of the DES prediction method in comparison with the STES e^2 and the STES Whybark predictor for all those datasets. We clearly observe that the DES prediction method outperforms the other two predictors. For two of the datasets the DES shows even more than 50% improvements for both of the other predictors.

Figure 7.12 below shows the improvements of the DES prediction method compared to the ES predictor with $\alpha = 0.5$, denoted as the ES(0.5) predictor. Despite the fact that DES is based on ES, the method shows a completely different accuracy: the difference between the RMSEs of the ES(0.5) predictions and the DES predictions ranges from -17% to +70% and differs for each dataset. Only for single dataset the ES(0.5) predicts significantly better than the DES. On average the DES shows 11% improvement.

Figure 7.13 illustrates the improvement of the DES prediction method in comparison with the NWS prediction method. The figure shows that DES mostly outperforms the NWS prediction method. For 6 of the 45 datasets the NWS prediction method is only 1 to 5 percent more accurate than the DES prediction method. However, on average 8% improvements can be performed by implementing the DES prediction method instead of the NWS prediction method.

Figure 7.14 illustrates the performance improvement of the DES prediction method in comparison with the AR(16) predictor. Figure 7.14 shows more fluctuations than Figure 7.13, because the DES and the NWS prediction methods have more similarities.

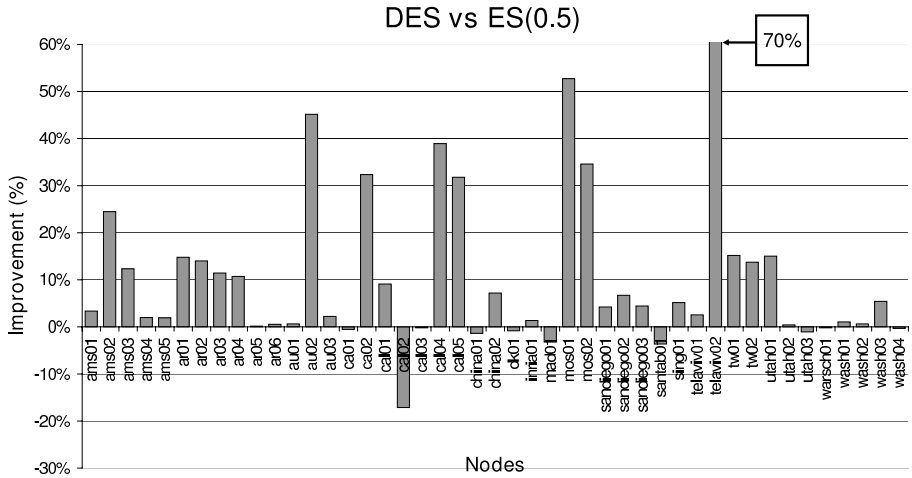


Figure 7.12: DES-predictor improvements compared to the ES(0.5) predictor

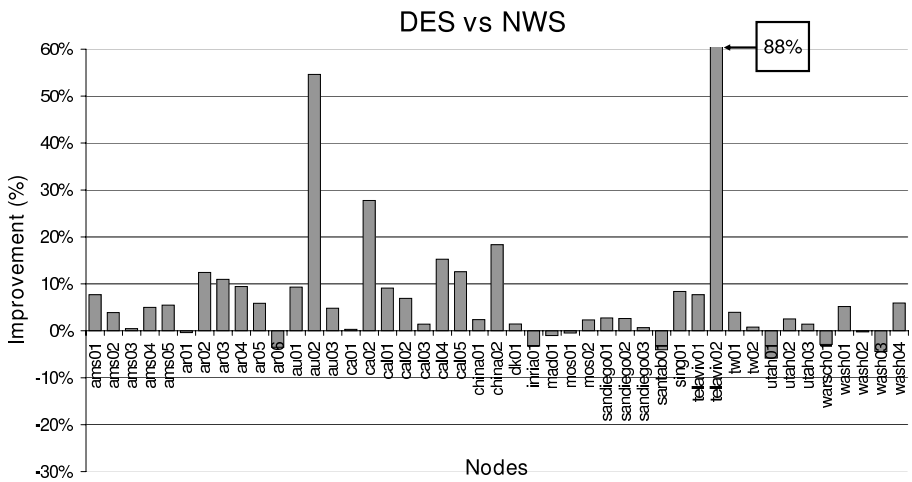


Figure 7.13: DES-predictor improvements compared to the NWS prediction method

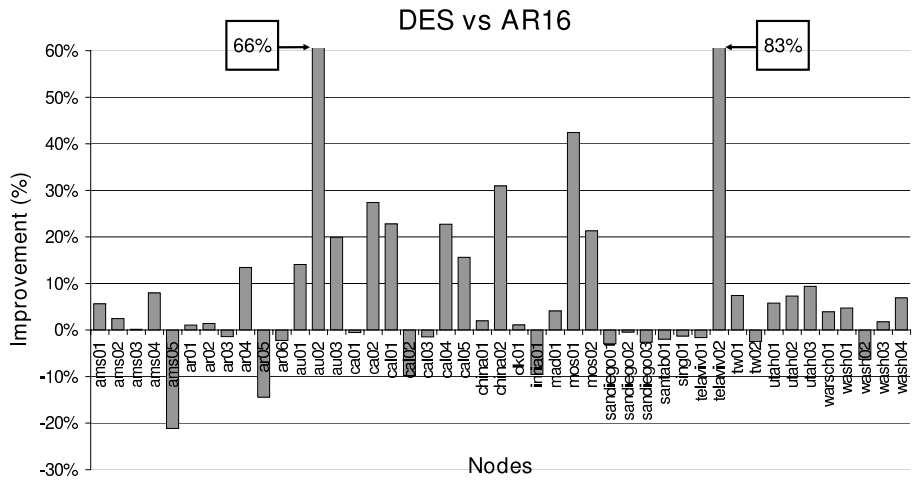
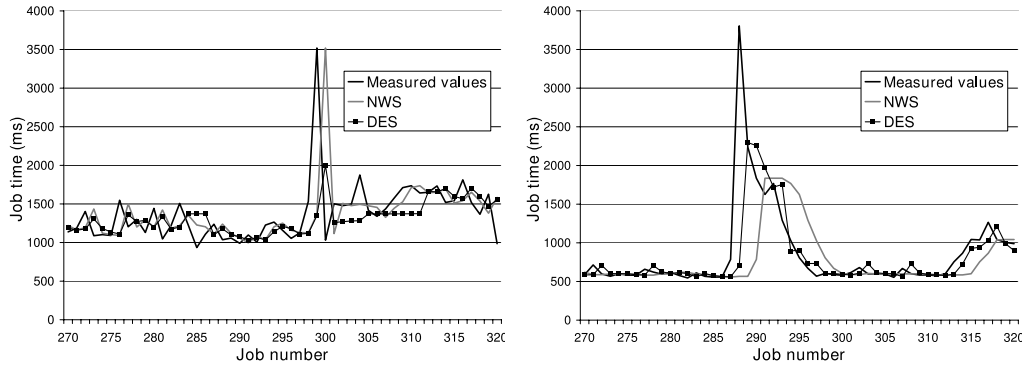


Figure 7.14: DES-predictor improvements compared to the AR16 predictor

For five datasets the AR(16) predictor shows more than 5% accuracy improvements compared to the DES prediction method. In fourteen cases the DES prediction method shows more than 10% improvement. For three datasets the DES prediction method is even more than 40% more accurate than the AR(16) predictor. On average, the DES prediction method is 9% more accurate than the AR(16). As we see in Figures 7.12 to 7.14 DES consistently outperforms all other predictors for datasets au02 and telaviv02. Further analysis show that the jumps in those datasets have a huge influence on the standard deviation. We found that the main reason for the enormous performance improvement is that the DES prediction method is the only method that reacts properly on the high jumps. Nevertheless, when we filter out the jumps with a deviation above ten times the standard deviation the DES still clearly outperforms the others predictors.

Next, we investigate the reason for the performance improvement of the DES prediction method in comparison with the best of the other predictors, the NWS prediction method. To this end, we analyze the situations in the datasets where DES clearly outperforms NWS. We observe two main reasons why DES outperforms the NWS prediction method: (1) DES reacts more properly on peaks, and (2) DES reacts more properly on level switches. To investigate this more carefully, we add the predictions of the DES prediction method in Figure 7.2. The results are graphically represented in Figure 7.15.

We notice the following from Figure 7.15(a). Due to the values before the jump, the NWS chooses a predictor that gives a high weight to the last value. When the jump appears, the NWS prediction for the next running time also gets very high, even though there is a high probability that it is a peak, because almost all the jumps in history were peaks and not level switches. The DES prediction method also gives a prediction that is slightly too high due to the fact that one jump in history did introduce a new level. However, as can be seen in Figure 7.15(a) at data points 300-312, the DES predictor sometimes shows a too stable pattern after a peak. In that case DES interprets every fluctuation as a jump without level switch and therefore does not react on the fluctuations. But, nevertheless, the DES prediction method is far more accurate than the NWS prediction method.



(a) **Peak on pre_ar02**

(b) **Level switch on pre_telaviv01**

Figure 7.15: DES- and NWS prediction method reacting on a peak and a level switch in the running times

In Figure 7.15(b), we have a different situation: jumps introduce a new level of running times. In this example we see that the NWS also shows a bad performance. Due to the fact that the values before the jumps need a predictor that has a more averaging property, the NWS also uses the average to predict the value after the jump despite the fact that the jumps in history very often introduced a new level. It takes three values (i.e., jobs durations) before the NWS prediction method realizes that the predictions are not accurate. The DES prediction method clearly shows a better pattern of predictions in the level switch.

Despite the two previous mentioned situations where the NWS prediction method does not react very properly on jumps, there are a lot of situations where the NWS prediction method does react appropriately. There are three situations where the NWS does react very accurately: (1) when a more averaging predictor is chosen because of the values before a jump, and it does not introduce a new level, (2) when a last value predictor is chosen before a jump and the jump introduces a new level, and (3) when the jumps appear in quick succession, the NWS still *remembers* what the best predictor was during the last jump, because jumps have a high impact on the RMSE, and therefore on the choice of the predictor.

To summarize, the comparison results show that in general the DES prediction method outperforms the existing predictions methods. Another well performing predictor for predicting running times of jobs is the NWS prediction method.

7.5 Correlation DES and statistics

The statistical characteristics of the resources in grid environments which are discussed in Chapter 3 differ for every grid environment. Consequently, the quality of the different predictors which are discussed in this chapter depends on those properties, and, therefore, it is hard to know in advance which predictor can be applied for which situation. In this section we investigate the relation between the statistical characteristics of the measurements and the quality of the predictors DES, NWS, and AR for these

measurements. In order to measure the quality difference between two predictors, we use the percentage improvement $\Delta\%(A, B)$ (see: (7.29)).

In addition we define the following statistics which potentially have a relation with the improvement percentage of DES compared to the NWS and the AR. The names of the quantities in the formulas and the abbreviations correspond to those of Chapter 3. The statistic $rmse_{jumps}/\sigma$ is the RMSE impact of the jumps divided by the standard deviation and indicates how much impact a single jump on average has on the standard deviation. Statistic $rmse_{jumps}/\mu$ is the RMSE impact of the jumps divided by the average. In addition, $rmse_{jumps}/(\mu * f_{jumps})$ equals the previous statistic divided by the fraction of jumps. This statistical property indicates the sum of the RMSEs of all jumps with their previous value, compared to the average of the dataset. Moreover, $rmse_{jumps}/(rmse_{lv} * f_{jumps})$ is the RMSE impact of the jumps divided by the RMSE of the last value and the fraction of jumps. This statistic tells something about the complete impact of all jumps together on the total amount of fluctuations. Next, the $mse_{jumps}/(\sigma^2 * f_{jumps})$ is the RMSE impact of the jumps divided by the variance and the fraction of jumps. This statistic has the property that it illustrates how much influence each single jump on average has on the variance, compared to the variance of the dataset. Finally, the $mse_{jumps}/(mse_{lv} * f_{jumps})$ corresponds to one of the above statistics. In order to compute this statistic, take the MSE impact of the jumps and divide it by the MSE of the last value and the fraction of jumps. This indicates how much influence each individual jump on average has on the variance, compared to the total fluctuations in the dataset. Table 7.2 represents the correlation coefficients between the statistics and the improvement percentages of DES compared to the NWS and the AR16 predictors.

We conclude from the investigations that many statistics have a positive relation with the improvement percentage of DES. The Coefficient of Variation has significant influence; the higher the standard deviation compared to the average, the higher the improvements that can be made by DES. Further, the standard deviation of the standard deviations divided by the average and the coefficient of variation of the standard deviations have a significant influence on the success factor. The RMSE impact of the jumps themselves does not have a high correlation with the improvement, but when the RMSE impact of the jumps is divided by the RMSE of the last value, the average, the average multiplied with the fraction of jumps, or the RMSE of the last value multiplied with the fraction of jumps, the impact is much higher. The same holds for the MSE impact of the jumps when it is divided by the variance multiplied with the fraction of jumps, and the MSE of the last value multiplied with the fraction of jumps. An increase of those above mentioned statistics leads to a bigger difference in the accuracy of the DES in comparison with the other predictors. We conclude that when at least one of the above mentioned statistics of a dataset has a high value, it is very useful to apply the DES predictor.

Furthermore, we observe the following. There are two statistics that have a slight negative impact on the accuracy of the DES predictor: the $\sigma/rmse_{lv}$ and the fraction of jumps. However, due to the fact that the correlation coefficient are not significantly negative, it is not possible to draw strong conclusions. The height of the rest of the statistical properties do not have a significant influence on the success of DES in comparison with NWS or AR16. We notice some difference between the results of the

Name	Correlation with improvement % DES vs. NWS	Correlation with improvement % DES vs. AR16
μ	0.03	-0.03
σ	0.07	-0.01
c	0.57	0.35
$rmse_{lv}$	0.12	0.04
$\frac{\sigma}{rmse_{lv}}$	-0.10	-0.21
$\sigma(\underline{\mu})$	0.03	-0.03
$c(\underline{\mu})$	0.36	0.23
$\frac{\sigma(\underline{\mu})}{\sigma}$	-0.06	0.00
$\sigma(\underline{\sigma})$	0.23	0.10
$\frac{\sigma(\underline{\sigma})}{\mu}$	0.85	0.63
$c(\underline{\sigma})$	0.83	0.66
f_{jumps}	-0.36	-0.19
σ_{jumps}	0.17	0.06
$rmse_{jumps}$	0.20	0.10
$\frac{\sigma_{jumps}}{\sigma}$	0.52	0.52
$\frac{rmse_{jumps}}{\sigma}$	0.33	0.41
$\frac{rmse_{jumps}}{rmse_{lv}}$	0.49	0.41
$\frac{rmse_{jumps}}{\mu}$	0.65	0.51
$\frac{rmse_{jumps}}{\mu * f_{jumps}}$	0.90	0.74
$\frac{rmse_{jumps}}{rmse_{lv} * f_{jumps}}$	0.73	0.63
mse_{jumps}	0.09	0.06
$\frac{mse_{jumps}}{\sigma^2}$	0.37	0.48
$\frac{mse_{jumps}}{\sigma^2 * f_{jumps}}$	0.93	0.81
$\frac{mse_{jumps}}{mse_{lv} * f_{jumps}}$	0.80	0.69

Table 7.2: Correlations between the gain of DES and 24 different statistical properties

NWS compared to the AR16. The correlations belonging to the analyses with the NWS are higher than those of the AR16.

7.6 Conclusions

The conclusions drawn in Chapter 3 about the grid properties has raised the need for the development of effective methods for prediction of running times that are able to deal with those characteristics. In this chapter, we have analyzed the performance of a variety of prediction methods. Our results show that none of these methods is well-suited for dealing with the specifics of running times in a grid environment, including the presence of level switches and sudden peaks. To this end, we have developed a new prediction method, called DES, that overcomes the shortcomings of the existing methods by properly reacting to level switches and peaks. The power of DES lays in the fact that it is able to deal with both peaks and level switches.

Extensive comparisons with a large number of datasets show that DES is a highly effective method for predicting running times of jobs on shared processors. DES consistently outperforms the common grid prediction methods, such as the NWS predictor, AR(16), and ES, and moreover, gives much more accurate predictions than the Adaptive Exponential Smoothing Predictors Trigg and Leach, Whybark, Mentzer, Pantazopoulos and Pappis, and four kinds of promising Smooth Transition Exponential Smoothing predictors. Consequently, the predictor has proved its robustness against the completely different characteristics of the job runtimes that are measured on heterogeneous nodes.

In addition, we investigated the relation between the improvements in accuracy of the DES predictor compared to the NWS, and the Autoregressive predictor and 24 different statistics. We conclude that the statistic of statistic MSE impact of the jumps divided by the variance multiplied with the fraction of jumps has the highest correlation coefficients (0.93 and 0.81) with the improvements percentages.

8

FURTHER RESEARCH

The results presented in this thesis lead to a number of challenges for further research. First, in the conducted research of this thesis we have focused on the dynamics of processing speeds. However, in data-intensive grid applications the uncertainties in the available amount of network capacity may have a more significant impact than fluctuations in the processor speeds. To this end, many efforts are being undertaken to control these changing network capacities (e.g., [19, 71]). It is a challenge in grid environments to integrate the methods that cope with the dynamics of the processors and of the network, such that applications are robust against every type of fluctuation.

In addition to the research which is carried out in Chapter 3 it is interesting to investigate the correlations between the job runtimes on different processors. An effective statistical method to quantify these relations is the Cross Correlation Function (CCF). We plan to compute the CCFs of the job runtimes on different nodes and make contour plots, which will provide insight in those correlations.

An interesting and important question in running dynamic load balancing (DLB) applications in a grid environment is to derive the optimal number of iterations between successive load re-balancing steps. In the experiments of Chapter 4, the DLB-actions were performed each $N = 10$ iterations. However, due to the random nature of the grid environment, one may expect that more efficient load balancing schemes can be achieved by allowing load balancing actions to be performed at any moment, according to some dynamic algorithm that optimally balances the "cost" of load balancing actions, which depends on the number of processors, the problem size, and the overhead of rescheduling load, and the benefits in quickly reacting to load changes. Moreover, we have performed experiments with the SOR application. SOR has a specific linear structure, which is shown in Section 2.3. The question arises to what extent the results presented in this chapter are applicable to other SPMD applications. In-depth analysis of parallel applications with a non-linear structure is a challenging topic for further research.

In Chapter 5, we present a model that describes the iteration of a parallel program based on job replication (JR) and which runs on homogeneous nodes. It would be a challenge to develop such a model for the case that a program runs on heterogeneous nodes. This means that the nodes in the grid environment all can have a different job-

runtime distribution. Consequently, similar to the analyses of this chapter, the characteristics and sensitivity of the JR speedups on heterogeneous nodes can be investigated, and more insight in the speedups can be obtained.

Finally, the results presented in Chapter 7, which analyzes prediction methods in grids, lead to a number of challenges for further research as well. First, there is room for refinement of the DES predictor. For the running times of jobs, the current choice of parameters for the heights of jumps (i.e. a difference between two consecutive values of more than two times the standard deviation is a jump, and more than ten times the standard deviation is a high jump) leads to significant improvements in the predictions. We expect that those parameters partly depend on the properties of the dataset. We plan to investigate whether it is possible to adapt these parameters to the statistical properties of the datasets. Consequently, this means that predictors for job runtimes which are measured on different nodes choose different values for the above mentioned parameters. Moreover, this strategy can be applied if the job size is adapted. For this case it is reasonable that the level switches and peaks change their shapes and therefore it is necessary that the parameters adjust to the new characteristics. Second, we aim to apply the DES prediction method for predicting other types of grid property measurements (e.g., latency, CPU utilizations) or in other large-scale grid environments. Finally, the ultimate goal of the development of effective prediction methods in the context of the computational grid is to decrease the effective running times of distributed applications by triggering effective load re-balancing actions. Therefore, the next step is to quantify the actual improvements that can be obtained in the running times of distributed applications, which addresses a challenging area for further study.

BIBLIOGRAPHY

[1] http://www.cs.vu.nl/das2.

[2] http://www.planet-lab.org.

[3] http://www.eurogrid.org.

[4] http://www.gridlab.org.

[5] I. Ahmad and Y.-K. Kwok. A new approach to scheduling parallel programs using task duplication. In *Proceedings of the 23th International Conference on Parallel Processing (ICPP '94)*, pages 47–51, North Carolina State University, NC, USA, August 15-19, 1994. CRC Press.

[6] A. H. Alhusaini, C. S. Raghavendra, and V. K. Prasanna. Run-time adaptation with resource co-allocation for grid environments. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS '01)*, pages 87–97, San Francisco, CA, April 23-27, 2001. IEEE Computer Society.

[7] G. Attiya and Y. Hamam. Two phase algorithm for load balancing in heterogeneous distributed systems. In *Proceedings of the 12th Euromicro conference on parallel, distributed and network-based processing (PDP '04)*, pages 434–439, A Coruna, Spain, February 11-13, 2004. IEEE Computer Society.

[8] R. Bajaj and D. P. Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):107–118, 2004.

[9] I. Banicescu and Z. Liu. Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes. In *Proceedings of the High Performance Computing Symposium (HPC '00)*, pages 122–129, Washington, DC, USA, April 16-20, 2000.

[10] I. Banicescu and V. Velusamy. Load balancing highly irregular computations with the adaptive factoring. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS '02) - Heterogeneous Computing Workshop (HCW '02), on CD-rom*. IEEE Computer Society, April 15-19, 2002.

[11] J. Beran. *Statistics for long-memory processes*. Chapman and Hall, New York, 1994.

[12] F. D. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of the 9th ACM/IEEE conference on Supercomputing (SC '96)*, number 39, Pittsburgh, PA, USA, November 17-22, 1996. ACM Press/IEEE Computer Society.

[13] R. L. Cariño and I. Banicescu. A load balancing tool for distributed parallel loops. In *Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments (CLADE '03)*, pages 39–46, Seattle, WA, USA, 2003. IEEE Computer Society.

[14] H. Casanova. On the harmfulness of redundant batch requests. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC '06)*, pages 255–266, Paris, France, June 19-23, 2006. IEEE Computer Society.

[15] L. Cheng and I. Marsic. Modeling and prediction of session throughput of constant bit rate streams in wireless data networks. In *Proceedings of the 2003 IEEE Wireless Communications and Networking Conference (WCNC '03)*, pages 1733–1741, New Orleans, USA, March 16-20, 2003.

[16] J. Y. Colin and P. Chretienne. C.p.m. scheduling with small communication delays and task duplication. *Operations Research*, 39(4):680–684, 1991.

[17] H. Dail, G. Obertelli, F. Berman, R. Wolski, and A. Grimshaw. Application-aware scheduling of a magnetohydrodynamics application in the legion metasystem. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW '00)*, pages 216–228, Cancun, Mexico, May 1, 2000. IEEE Computer Society.

[18] S. Darbha and D. P. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 9(1):87–95, 1998.

[19] M. den Burger, T. Kielmann, and H. E. Bal. Balanced multicasting: High-throughput communication for grid applications. In *Proceedings of the 18th ACM/IEEE Conference on Supercomputing (SC '05)*, number 47, Seattle, WA, USA, November 12-18 2005. ACM Press/IEEE Computer Society.

[20] P. A. Dinda. The statistical properties of host load. *Scientific Programming*, 7(3-4):211–229, 1999.

[21] P. A. Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3):225–236, July 2002.

[22] P. A. Dinda. A prediction-based real-time scheduling advisor. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS '02)*, pages 35–42, Fort Lauderdale, FL, USA, April 15-19, 2002. IEEE Computer Society.

[23] P. A. Dinda and D. R. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4):265–280, 2000.

[24] A. M. Dobber, G. M. Koole, R. Righter, and R. D. van der Mei. Scheduling jobs on homogeneous processors. In preparation.

[25] A. M. Dobber, G. M. Koole, and R. D. van der Mei. Dynamic load balancing for a grid application. In *Proceedings of the 11th International Conference on High Performance Computing (HiPC '04)*, pages 342–352, Bangalore, India, December 19-22, 2004. Springer-Verslag.

[26] A. M. Dobber, G. M. Koole, and R. D. van der Mei. Dynamic load balancing experiments in a grid. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, pages 123–130, Cardiff, Wales, UK, May 9-12, 2005. IEEE Computer Society.

[27] A. M. Dobber, R. D. van der Mei, and G. M. Koole. Dynamic load balancing and replicating in a global-scale grid enviroment: a comparison. Submitted.

[28] A. M. Dobber, R. D. van der Mei, and G. M. Koole. Prediction method for job runtimes on shared processors: Survey, statistical analysis and new avenues. In revision for *Performance Evaluation*.

[29] A. M. Dobber, R. D. van der Mei, and G. M. Koole. Effective prediction of job processing times in a large-scale grid environment. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC '06) (poster)*, pages 359–360, Paris, France, June 19-23, 2006. IEEE Computer Society.

[30] A. M. Dobber, R. D. van der Mei, and G. M. Koole. Statistical properties of task running times in a global-scale grid environment. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '06)*, pages 150–153, Singapore, May 16-19, 2006. IEEE Computer Society.

[31] D. J. Evans. Parallel SOR iterative methods. *Parallel Computing*, 1:3–18, 1984.

[32] R. Fildes. Quantitative forecasting-the state of the art: extrapolative models. *Journal of the Operational Research Society*, 30:691–710, 1979.

[33] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

[34] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[35] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, 1998.

[36] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, 2003.

[37] G. C. Fox. An application perspective on high-performance computing and communications. Technical Report SCCS-757, Syracuse University Technical, April 1996.

[38] A. Giersch, Y. Robert, and F. Vivien. Scheduling tasks sharing files on heterogeneous master-slave platforms. In *Proceedings of the 12th Euromicro conference on parallel, distributed and network-based processing (PDP '04)*, pages 364–371, A Coruna, Spain, February 11-13, 2004. IEEE Computer Society.

[39] L. Guodong, C. Daoxu, D. Wang, and Z. Defu. Task clustering and scheduling to multiprocessors with duplication. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS '03)*, number 6b, Nice, France, April 22-26, 2003. IEEE Computer Society.

[40] L. A. Hageman and D. M. Young. *Applied Iterative Methods*. Academic Press, 1981.

[41] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS '96)*, pages 13–24, New York, NY, USA, 1996. ACM Press.

[42] S. F. Hummel, E. Schonberg, and L. E. Flynn. Factoring: A method for scheduling parallel loops. *Communications of the ACM archive*, 35(8):90–101, 1992.

[43] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):280–289, 1977.

[44] H. D. Karatza and R. C. Hilzer. Parallel and distributed systems: Load sharing in heterogeneous distributed systems. In *Proceedings of the 34th conference on Winter Simulation: exploring new frontiers*, pages 489–496, San Diego, CA, USA, December 08-11, 2002. ACM Press.

[45] J. F. Kenney and E. S. Keeping. *Mathematics of Statistics*, chapter 15: Linear Regression and Correlation, pages 252–285. van Nostrand, 3 edition, 1962.

[46] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien. Characterizing and evaluating desktop grids: An empirical study. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, number 26b, Santa Fe, NM, USA, April 26-30, 2004. IEEE Computer Society.

[47] G. M. Koole and R. Righter. Resource allocation in grid computing. *Journal of Scheduling*.

[48] Y.-K. Kwok. Parallel program execution on a heterogeneous pc cluster using task duplication. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW '00)*, pages 364–374, Cancun, Mexico, May 1, 2000. IEEE Computer Society.

[49] J. H. Lambert. Observations variae in mathesin puram. *Acta Helvitica, physico-mathematico-anatomico-botanico-medica*, 3:128–168, 1758.

[50] B. Lee and J. Schopf. Run-time prediction of parallel applications on shared environments, poster paper. In *Proceedings of the 5th IEEE International Conference on Cluster Computing (Cluster '03) (poster)*, pages 487–491, Kowloon, Hong Kong, China, December 1-4, 2003. IEEE Computer Society.

[51] W. Leland and T. J. Ott. Load-balancing heuristics and process behavior. In *Proceedings of the 1986 ACM SIGMETRICS joint international conference on Computer performance modelling, measurement and evaluation (SIGMETRICS '86/PERFORMANCE '86)*, pages 54–69, Raleigh, NC, United States, 1986. ACM Press.

[52] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02)*, pages 63–75, Edinburgh, Scotland, UK, 2002. IEEE Computer Society.

[53] V. M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Transactions on computers*, 37(11):1384–1397, 1988.

[54] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante. Characterizing and predicting tcp throughput on the wide area network. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS '05)*, pages 414–424, Columbus, OH, USA, June 6-10, 2005. IEEE Computer Society.

[55] J. T. Mentzer. Forecasting with adaptive extended exponential smoothing. *Journal of the Academy of Marketing Science*, 16:62–70, 1988.

[56] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Performance Evaluation*, 12(4):269–284, 1991.

[57] Z. Nemeth, G. Gombas, and Z. Balaton. Performance evaluation on grids: Directions, issues and open problems. In *Proceedings of the 12th Euromicro conference on parallel, distributed and network-based processing (PDP '04)*, pages 290–297, A Coruna, Spain, February 11-13, 2004. IEEE Computer Society.

[58] S. N. Pantazopoulos and C. P. Pappis. A new adaptive method for extrapolative forecasting algorithms. *European Journal of Operational Research*, 94:106–111, 1996.

[59] G.-L. Park, B. Shirazi, and J. Marquis. Mapping of parallel tasks to multiprocessors with duplication. In *Proceedings of the 31st annual Hawaii International Conference on System Sciences (HICSS '98)*, volume 7, pages 96–105, Kohala Coast, Hawaii, USA, January 6-9, 1998. IEEE Computer Society.

[60] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[61] Y. Qiao and P. Dinda. Network traffic analysis, classification, and prediction. Technical Report NWU-CS-02-11, Department of Computer Science, Northwestern University, January 2003.

[62] J. Schopf and F. Berman. Performance prediction in production environments. In *Proceedings of the 12th International Parallel Processing Symposium / 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP '98)*, pages 647–653, Orlando, Florida, USA, March 30 - April 3, 1998. IEEE Computer Society.

[63] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society, 1995.

[64] K. H. Shum. Adaptive distributed computing through competition. In *Proceedings of the 3th IEEE International Conference on Configurable Distributed Systems (CDS '96)*, pages 200–227, Annapolis, MD, USA, May 6-8, 1996. IEEE Computer Society.

[65] T. Tabirca, S. Tabirca, L. Freeman, and L. T. Yang. A static workload balance scheduling algorithm. In *Proceedings of the 31st International Conference on Parallel Processing Workshops (ICPP '02 Workshops)*, pages 235–239, Vancouver, BC, Canada, August 20-23, 2002. IEEE Computer Society.

[66] J. W. Taylor. Smooth transition exponential smoothing. *Journal of Forecasting*, 23:385–404, 2004.

[67] D. W. Trigg and A. G. Leach. Exponential smoothing with an adaptive response rate. *Operations Research Quarterly*, 18:53–59, 1967.

[68] J. W. Tukey. *Exploratory Data Analysis*, pages 39–43. Addison-Wesley, Reading, MA, 1977.

[69] Y. T. Wang and R. J. T. Morris. Load sharing in distributed systems. *IEEE Transactions on computers*, pages 204–217, 1985.

[70] D. C. Whybark. Comparison of adaptive forecasting techniques. *Logistics Transportation Review*, 8:13–26, 1973.

[71] R. Wolski. Forecasting network performance to support dynamic scheduling using the Network Weather Service. In *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing (HPDC '97)*, pages 316–325, Portland, OR, USA, August 5-8, 1997. IEEE Conputer Society.

[72] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Performance Evaluation Review*, 30(4):41–49, 2003.

[73] R. Wolski, N. T. Spring, and J. Hayes. Predicting the CPU availability of time-shared unix systems on the computational grid. *Cluster Computing*, 3(4):293–301, 2000.

[74] L. Yang, I. Foster, and J. M. Schopf. Homeostatic and tendency-based cpu load predictions. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS '03)*, pages 42–50, Nice, France, April 22-26, 2003. IEEE Computer Society.

[75] D. York. Least-square fitting of a straight line. *Canadian Journal of Physics*, 44:1079–1086, 1966.

[76] M. J. Zaki, W. Li, and S. Parthasarathy. Customized dynamic load balancing for a network of workstations. *Journal of Parallel and Distributed Computing*, 43(2):156–162, 1997.

SAMENVATTING

Robuuste applicaties in gedeelde verspreide systemen

De laatste decennia hebben bedrijven en instellingen een groeiende behoefte aan computerkracht. Het KNMI is een voorbeeld van zo'n instelling omdat zij iedere dag grote en complexe berekeningen uitvoert om de weersvoorspellingen voor de volgende dag te berekenen. Het KNMI verbetert ieder jaar haar modellen en heeft daarvoor steeds meer computerkracht nodig. Ook financiële instellingen, die iedere dag de actuele situatie berekenen van al hun klanten, voeren grote berekeningen uit. Door de groeiende hoeveelheid gegevens die bijgehouden wordt door deze instellingen groeit ook hun vraag naar computerkracht. Een ander toepassingsgebied is de veiligheid op bijvoorbeeld luchthavens: binnen enkele jaren worden irisscans en vingerafdrukken van alle passagiers ter plaatse vergeleken met die van wereldwijde databases. Daarbij is een vereiste dat de applicatie binnen enkele seconden aangeeft of de passagier geweigerd moet worden. Hieraan kan alleen voldaan worden als er genoeg rekenkracht beschikbaar is.

Op dit moment voorziet men in de vraag naar computerkracht door per type toepassing tientallen computers aan elkaar te koppelen in *clusters* zodat deze applicaties hierop uitgevoerd kunnen worden. Deze lokale netwerken van computers zijn kostbaar en moeten vaak worden uitgebreid of er dienen computers vervangen te worden om aan de groeiende behoefte aan computerkracht te blijven voldoen.

Tegelijkertijd is het aantal computers in de wereld erg hoog en neemt jaarlijks enorm toe. Bovendien worden zij door nieuwe processortechnologieën steeds sneller. De bezettingsgraad op deze computers is over het algemeen laag, wat betekent dat de capaciteit vaak niet volledig of helemaal niet wordt gebruikt. Door de ontwikkeling van hogesnelheidsverbindingen kunnen deze computers aan elkaar gekoppeld worden zodat er een groeiende, wereldwijde bron van computerkracht, genaamd een *Grid*, ontstaat. Dit Grid is een goed alternatief voor de hierboven genoemde clusters door zijn grote en steeds groeiende beschikbare capaciteit die ontstaat door de schaalvoordelen.

Een Grid heeft dus voordelen ten opzichte van een cluster van computers, maar er zijn een aantal punten waar aandacht aan geschonken dient te worden. Zo dient ten eerste de beveiliging sterk verbeterd te worden. Het aantal mogelijkheden om bij een computer te kunnen inbreken zal toenemen. Daarnaast moeten alle programma's op een dusdanige manier geschreven worden dat de berekeningen op iedere computer uitgevoerd kunnen worden. Bovendien moet er een geavanceerd systeem komen dat aan de ene kant alle berekeningen ontvangt en ze aan de andere kant zo goed mogelijk

distribueert in het Grid. Ook moet er rekening gehouden worden met het feit dat het aantal beschikbare computers voortdurend verandert.

De toewijzing van de beschikbare capaciteit in een cluster of een Grid kan op twee manieren plaatsvinden: volgens (1) reservering van de volledige capaciteit: de gebruikers reserveren de volledige capaciteit van een aantal processoren en wachten tot deze vrij zijn, en (2) *processor sharing*: de capaciteiten van de processoren worden op hetzelfde moment door verschillende gebruikers gedeeld. Wanneer het aantal gebruikers in het Grid op een moment lager is of iets hoger dan het aantal beschikbare computers dan is reservering van de gehele capaciteit beter omdat de wachttijd voordat de berekeningen kunnen beginnen vrij laag is en vervolgens zal het uitvoeren van de berekeningen minder tijd vergen omdat men de processoren maximaal kan inzetten. Echter, een hoger aantal gebruikers leidt bij dit reserveringstype tot lagere efficiëntie van de processoren en significant langere wachttijden. In dit geval is processor sharing een veelbelovend alternatief: bij dit reserveringstype zijn er per definitie nauwelijks wachttijden voordat de berekeningen kunnen beginnen. Echter, de rekentijden zijn vaak langer en fluctueren voortdurend door de veranderende vraag naar processorcapaciteit.

Op dit moment maken clusters over het algemeen gebruik van volledige capaciteitsreservering en de wereldwijde grids van processor sharing door het grotere aantal gebruikers op deze grids. Om de wereldwijd verspreide *processor-sharing* grids een goed alternatief te laten zijn voor de clusters, die gebaseerd zijn op volledige capaciteitsreservering, moeten de applicaties robuust gemaakt worden tegen het veranderende aantal beschikbare computers en de steeds fluctuerende snelheden van de processoren. In dit proefschrift richten we ons op *de ontwikkeling van methoden om grid applicaties robuust te maken tegen de voortdurend veranderende effectieve snelheden van de processoren.*

Een parallel programma dat berekeningen stuurt naar meerdere computers en vervolgens moet wachten op alle uitkomsten zal door deze fluctuaties veel vertraging oplopen. Er zijn over het algemeen twee typen implementaties die gebruikt worden om hierop te anticiperen: (1) Dynamische Load Balancing (DLB) en (2) Job Replicatie (JR). DLB past de groottes van de berekeningen dynamisch aan aan de voorspelde snelheden van de processoren. JR repliceert de berekeningen een aantal keer, verstuurt ze naar verschillende processoren en wacht totdat één van de processoren klaar is. Op dat moment worden de nog niet afgeronde gerepliceerde berekeningen beëindigd en wordt een nieuwe serie van berekeningen verstuurd. JR maakt geen gebruik van voorspellingen van de processorsnelheden.

In dit proefschrift behandelen we de volgende onderwerpen. In hoofdstuk 2 beschrijven we de gebruikte experimentele grid omgeving, Planetlab [2], en de gebruikte applicatie, SOR [31]. Bovendien worden diverse termen of begrippen gedefinieerd, die in het hele proefschrift worden gebruikt en worden de simulaties van JR en DLB uitgebreid omschreven.

In hoofdstuk 3 zijn de statistische eigenschappen van berekentijden onderzocht. In dit hoofdstuk concluderen we dat de karakteristieken van deze tijden per processor sterk verschillen en dat er veel niveausprongen en pieken in de rekentijden zitten.

Verder is in hoofdstuk 4 onderzocht hoe verschillende DLB implementaties in wereldwijd verspreide grid-omgevingen draaien. Voor dit onderzoek zijn simulaties en experimenten met een DLB implementatie uitgevoerd. De conclusie is dat DLB

een effectieve methode is om in te spelen op veranderingen in de rekentijden en dat de winst toeneemt naarmate grid applicaties meer processoren gebruiken.

Daarnaast zijn in hoofdstuk 5 diverse modellen ontwikkeld om schattingen te maken van de rekentijden van parallelle programma's op homogene processoren die gebruik maken van JR. Bovendien zijn er simulaties van JR implementaties op homogene en op heterogene processoren uitgevoerd. Hiermee schatten we de winst van JR ten opzichte van bestaande implementaties. Deze analyses laten zien dat ook JR een effectieve methode is om in te spelen op een dynamische omgeving.

In hoofdstuk 6 vergelijken we de resultaten van DLB en JR voor verschillende cases. De conclusie is dat DLB en JR effectiever zijn naar mate er meer en hogere fluctuaties zijn. DLB verdient de voorkeur boven JR wanneer er goede voorspellingen van de processorsnelheden te maken zijn, anders verdient JR de voorkeur. Vervolgens introduceren en motiveren we een eenvoudig te meten statistiek waarbij geldt dat als deze groter is dan een te bepalen drempelwaarde dat DLB effectiever is en anders JR. Dit vormt de basis voor een adaptieve methode die dynamisch kiest tussen een DLB en een JR methode. Deze adaptieve methode presteert in alle situaties op zijn minst beter als één van de twee enkele methoden.

In hoofdstuk 7 richten we ons vervolgens op het ontwikkelen van een nieuwe voorspelmethode voor rekentijden op gedeelde processoren. Hiervoor zijn de sterke en zwakke punten van bestaande methoden geanalyseerd op basis van een theoretische en een statistische analyse. De conclusie van deze analyse is dat de bestaande methoden niet goed omgaan met de vele niveausprongen en pieken in de rekentijden. Al het verkregen inzicht bij diverse analyses is gebruikt om een nieuwe voorspelmethode te ontwikkelen. Een vergelijking van de nauwkeurigheid van de voorspellers laat zien dat de nieuwe voorspeller een verbetering is ten opzichte van de bestaande methoden.

Uiteindelijk worden in hoofdstuk 8 een aantal mogelijkheden voor verder onderzoek aangegeven.

ABOUT THE AUTHOR

Anton Menno Dobber was born in Naarden, The Netherlands, on July 9, 1980. He graduated from Grammar School at De Meergronden, Almere, in June 1998. In December 2002 he received his master's degree in Business Mathematics and Informatics (cum laude) from the Vrije Universiteit in Amsterdam. During his study, he is recipient of the Software Engineering award 1999 and the BMI Thesis award 2002. He did his internship at CMG and wrote a thesis on the topic "Basel 2: Operational Risk". Subsequently, he started his PhD research project at the research group Optimization of Business Processes at the department of Mathematics at the Vrije Universiteit in Amsterdam. Menno defends his PhD thesis at the Vrije Universiteit Amsterdam on November 13, 2006.