Workload Modeling for Parallel Computers

Baiyi Song

Robotic Research Institute

Section Information Technology

The University of Dortmund

A thesis submitted for the degree of

Doktor-Ingenieur

Acknowledgements

I want to thank all those who made this work possible by their supports. In particular, I want to express my appreciation to my doctorial advisor, Prof. Dr.-Ing. Uwe Schwiegelshohn, for the source of instructive guidance and supports over the past years.

I am grateful to all people at Robotic Research Institute at the University of Dortmund for the excellent working atmosphere. Especially, I would also like to thank Dr.-Ing. Ramin Yahyapour, whose efforts and expertise help me to finish my study. I would like to express my gratitude to Dipl.-Inform. Carsten Ernemann, who give me many helpful suggestions and supports. I would also like to thank the Graduate School of Production Engineering and Logistics at the University of Dortmund, which provides me not only the financial supports but also the broad contacts with other departments. I would like to thank many of my chinese friends with them I had a pleasure time in Dortmund.

At last, I would like to thank my parents, my wife and my son, who always encourage me with their love, belief and patience.

Abstract

The availability of good workload models is essential for the design and analysis of parallel computer systems. A workload model can be applied directly in an experimental or simulation environment to verify new scheduling policies or strategies. Moreover, it can be used for extrapolating and predicting future workload conditions. In this work, we focus on the workload modeling for parallel computers. To this end, we start with an examination of the overall features of the available workloads. Here, we find a strong sequential dependency in the submission series of computational jobs. Next, a new approach using Markov chains is proposed that is capable of describing the temporal dependency. Second, we analyze the missing attributes in some workloads. Our results show that the missing information can be still recovered when the relevant model is trained from other complete data set. Based on the results of overall workload analysis, we begin to inspect the workload characteristics based on particular user-level features. That is, we analyze in detail how the individual users use parallel computers. In particular, we cluster the users into several manageable groups, while each of these groups has distinct features. These different groups provide a clear explanation for the global characteristics of workloads. Afterwards, we examine the user feedbacks and present a novel method to identify them. These evidences indicate that some users have an adaptive tendency and a complete workload model should not ignore the users' feedbacks. The work ends with a brief conclusion on the discussed modeling aspects and gives an outlook on future work.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

AIC	Akaike's Information Criteria
ARIMA	Auto-Regressive Integrated Moving Average
CLARA	Clustering Large Applications
FCFS	First Come First Service
HMM	Hidden Markov Model
KS	Kolmogorov-Smirnov
MCM	Markov Chain Model
MPI	Message Passing Interface
MPPs	Massively Parallel Processors
MUGM	Mixed User Group Model
PAM	Partition Around Medoids
PDM	Probabilistic Distribution Model
PVM	Parallel Virtual Machine
SA	Squashed Area
SIQR	Semi-Inter Quartile Range
SVM	Support Vector Machine
VLMC	Variable Length Markov Chain

Symbols

γ,β	The parameters of Gamma distribution
a	The dimension of Markov chain for parallelism
b	The dimension of Markov chain for runtime
cor	Pearson's correlation coefficient
D	The set of jobs
d_i	The parameter set for job i
d_i^p	The parallelism of job i
d_i^r	The runtime of job i
d_i^u	The ownership of the job i
J	The number of users in the system
K	Users are clustered into K groups
p, q	The parameters of Beta distribution

Chapter 1

Introduction

1.1 Motivation

Many traditional science disciplines as well as recent multimedia applications are increasingly dependent upon powerful high-performance systems for the execution of both computationally intensive and data intensive simulations of mathematical models and their visualizations. Parallel computing in particular has emerged as an indispensable tool for problem solving in many scientific domains during the course of the past fifteen years, e.g., weather forecasting, climate research, molecular modeling, physics simulations.

A parallel computer is a high-end machine designed to support the execution of parallel computational jobs. It can be composed of hundreds of high-speed processors, often called nodes. The processors are interconnected by a very fast network. A variety of parallel computers have been developed and are available to the user community. This variety ranges from the traditional Massively Parallel Processors (MPPs), to distributed shared memory systems, to clusters or networks of stand-alone workstations or PCs, to even geographically dispersed meta-systems or Grids connected by high-speed Internet connections. Research and development efforts focus on building faster processors, more powerful memories at all hierarchy levels, and on building fast networks with higher bandwidth and lower latencies. All these efforts contributed to the broad deployment of high-performance parallel systems.

Complementary to the hardware advance is the availability of transparent, highly portable, and robust software environments like Message Passing Interface (MPI) and Parallel Virtual Machine (PVM). Such environments hide the architecture details from the end-users and contribute to the portability and robustness of parallel jobs across a variety of hardware substrates. The availability of such environments transforms every intranet into a high performance system, thus increasing the user access to parallel systems. Recently, the concept of Grid computing has been promoted [25, 26]. Grid computing provides shared access to a potentially large number of geographically distributed heterogeneous computational resources which are made available by independent providers. Such Grids are used to solve large scale problems, which are otherwise intractable due to their diverse requirements in terms of computing power, memory and storage.

The availability of different parallel systems as well as the diversity of available hardware and software make the arbitration and management of resources among the user community a non-trivial problem. For example, a number of users typically attempt to use the system simultaneously; the requests of resources are variable in the parallelism of the applications and their respective computational and storage needs; sometimes execution deadlines must be met. Therefore, efficient scheduling systems are required to manage parallel computer resources. It is the task of the scheduling system to resolve the resource conflicts between the different jobs that are submitted to the system. It has to meet users' specifications and fulfill owners' requirements. A typical task of a scheduling system is , e.g., allocating resources according to the user's requirements (i.e., cater to the interests of a user) and maximize the system throughout (i.e., maximize system utilization, which is particularly important to amortize the cost of parallel computers). To this end, the scheduling system has to decide when to allocate resources for a particular job and when to delay a job in favor of executing others.

Here, workload modeling plays a vital role in designing scheduling systems. Since most parallel computers are very expensive machines, conducting extensive experiments on an actual installation to select suitable algorithms or testify new scheduling strategies is rarely an option. Instead, simulations are often executed to analyze new strategies. Therefore, a suitable realistic workload model is required that can be used for the simulation. It helps to compare different scheduling algorithms and explore the performance of the system in a multitude of scenarios. Because the performance of a system can only be interpreted and compared correctly with respect to the processed load, workload modeling, i.e. selecting and characterizing the load, is a central issue in performance evaluation.

Open Problems in Workload Modeling

The research presented in this work has been focused on workload modeling for parallel computers. Workload modeling has been subject to research for a long time, not only in the field of parallel computers but also in many other applications, e.g., web server characterization, network traffic description. Reviewing the available literature about workload modeling, we found that the workload models can be generally classified into three classes according to the number of users in the applications:

(a) *A large number of* independent users (say, thousands of individuals, or even more) contribute to a workload, e.g., in the field of telecommunication, a probabilistic distribution model [28, 41, 43, 55] normally works because the workloads from many independent users usually can be regarded as samples from a certain classical distribution, like Gaussian or Poisson.

(b) Only *a few* users (say, less than 10 users) are the contributors of a workload, like peer to peer Computing and special chip design [6, 10], specific models are required to describe each individual user or workload. That is, every user is represented using either a different model or a different parameter setting.

(c) *A medium number of* users (say, hundreds of users) generate a workload. Currently, the methods from class (a) and (b) are used, i.e., a general statistical model and a set of user-specific models [30, 35].

Since the user community of a parallel computer is medium, i.e. hundreds of users [23, 48, 49, 50], workload modeling for parallel computer belongs to the class (c). However, neither a general distribution model from class (a) nor a set of user-specific models from class (b) can work in this case. It is mainly because:

- A probabilistic distribution model is based on the assumption of independent sampling. However, when the size of user community is medium (hundreds of users), some users' patterns may still be observed in the final mixed workload. Therefore, the assumption of independent sampling may not hold any more and the retained patterns tend to be ignored by a distribution model.

- A general model describes the global characteristics of a workload. However, it does not consider individual user behaviors. Therefore, it can not provide a clear explanation to many phenomena in overall workloads from a user point of view.

- Due to the medium size of the user community, it is infeasible to apply a specific model for each individual user. Otherwise, the number of parameters will be too large and the scalability of the model will be lost.

Therefore, a more suitable model is required to analyze and characterize the workload of parallel computers. This is the focus of our work. We will propose a new model to address the complex behavior of users, to generalize similar submission behaviors and to consider the users' feedback behaviors. More details will be given in the next section.

Contribution of the Work

The objectives of our new workload model are to provide an adequate representation of the workload and meanwhile to characterize the way that users interact with parallel computers. The presented work serves on the one hand as a basis for deriving new workload models, and on the other hand as a beneficial supplement to existing approaches as new scheduling systems for parallel computers may include such models to predict the future workload situation. Several important aspects are considered in our new workload model:

- We inspect the temporal relations between jobs. A new approach is proposed to address the temporal relation in job series of workloads.
- Some attributes in the available workloads are missing due to the different resource configurations. Here, we propose a parameterized distribution model to describe the relation between missing and existing attributes.
- We put forward to a novel method to cluster heterogenous users into groups, while each of these groups has distinct features. Thus, more complicated scenarios can be simulated for evaluations by adjusting the parameters of user groups.
- We introduce implicit influential factors as representatives to examine the users' feedback behaviors. A linear model is used to model the feedbacks. With the model, the feedbacks can be identified and represented by a few parameters.

1.2 Organization of the Thesis

This thesis is organized as follows. Following this introduction, we discuss in Chapter 2 the details of workload modeling for parallel computers. Some essentials about performance evaluation using workload models are introduced and the traditional methods are presented. Based on the comparison of the existing approaches, a novel model structure is proposed.

Chapter 3 is devoted to modeling temporal relations in overall workloads. We will describe the details of our new method. That is, two correlated Markov chains are proposed to depict the temporal relations and the parameter correlations. Before turning to the next chapter, the comparisons of static and dynamic characteristics are made to verify our correlated Markov chains method.

Chapter 4 deals with missing information in the existing workloads. Not all available workloads provide the same set of information needed for some scheduling systems. Here, we take estimated runtime as an example to explain how the missing information is analyzed and modeled. The difference between estimated and real runtime is explained and then a parameterized distribution is given to model the estimated runtime, which is missing in some workloads.

In Chapter 5, the method to characterize individual users is given. The challenges to model the individual submission behaviors are discussed. Next, a user-group based workload model is given and the detailed steps to construct the model are introduced and corresponding results are presented.

Chapter 6 discusses the users' feedback behaviors. First of all, several implicit influential factors are introduced and then a descriptive model based on linear regression is proposed. Afterwards, the details of feedbacks identifications are given. The potential reason and implication of the feedbacks are discussed as well.

In Chapter 7, we demonstrate how these different modeling aspects can be combined and give the future direction about our research work. Several optional methodologies and models are discussed. To take an example, we explain how a new model is constructed by the combination of temporal relations and user groups. Finally, the dissertation ends with a brief conclusion.

Chapter 2

Workload Modeling for Parallel Computers

Modeling, i.e. analyzing and characterizing workloads, is an important task in designing scheduling systems for parallel computers, as the estimated or observed performance results depend on the characteristics of workloads. In this chapter, we will give a detailed description of the workload modeling problem in the field of parallel computers. Based on a broad overview of relevant work, a new structure is proposed. It is a collection of several fundamental components to address different aspects of workloads. Our new model can be adapted to meet particular situations given by the goals of specific evaluation study.

First of all, we shall give an explanation of a parallel environment which is considered in our work.

2.1 Parallel Computing Environment

Parallel Architecture

As we have mentioned, a parallel computer is a high-end machine, which is used to support the execution of parallel computational tasks. It is usually composed of hundreds of high-speed processors or nodes, which are interconnected by a very fast network. There are many different kinds of parallel computers (or ”parallel processors”). They are distinguished by the kind of interconnection between processors (known as ”processing elements” or PEs) and between memories. In our work, we assume that a parallel system is composed of identical processors or nodes. This coheres with the observation that many large scale systems for computational purposes consist of predominantly homogeneous partitions [19].

Job Description

A parallel computer is used for running computational tasks. Typically, these tasks use a certain amount of processors for a period of time. In our work, such a task is referred to as a *job*. A rectangle can be used to represent a job, with its *width* for *parallelism* and its *length* for the runtime as shown in Figure 2.1. Here, we use *parallelism* to refer to the number of processors or nodes used by a job (as shorthand for "number of nodes", or "degree of parallelism") and *runtime* for the span of time starting when a job commences execution and ending when it terminates (otherwise known as "duration" or "lifetime"). The product of runtime and parallelism of a job, which represents the total resource consumption (in CPU-seconds) used by the job, is called its *squashed area*. The term *workload* is referred to the data set recording historical job submissions. The term *workload model* is referred to the statistical model to describe the real workloads.

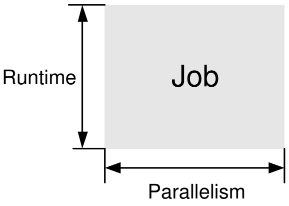

Figure 2.1: A job is represented by a rectangle in our study.

Scheduling Perspective

The scheduling problem of parallel computers is the composite problem of deciding *where* and *when* a job should execute. As we show in Figure 2.2, a scheduling system has to decide on which nodes (also indicated as the processor allocation or job scheduling problem) and in what order (also indicated as the process dispatching problem) the job will run. In our work, we consider *space-sharing* instead of *time-sharing* scheduling strategy, which is widely adopted by many parallel systems. Space-sharing scheduling restricts that two jobs executing concurrently must be disjoint. When a job is started on a machine, it runs to its end or is terminated. In our study, we specify that a job can not be stopped or interrupted unless it is finished, since many scheduling systems also follow this rule.

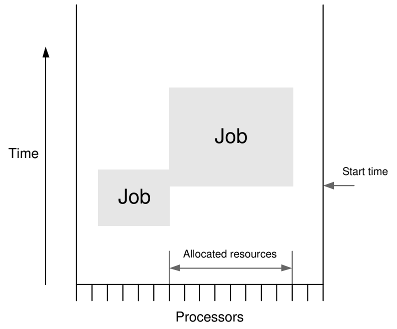

Figure 2.2: A job is scheduled by a scheduling system.

User Perspective

A medium number of users attempt to use parallel machine simultaneously, as it is shown in Figure 2.3. They submit jobs to the scheduling system and specify the details of resource requirements, including the number of processors, runtime, as well as some specific requirements. Usually, users make submissions from time to time. The scheduling system has no direct knowledge about the users' next submissions. This is the typical *online* scenario. Users have their own object functions and their future submissions may be affected by their satisfaction with the parallel system.

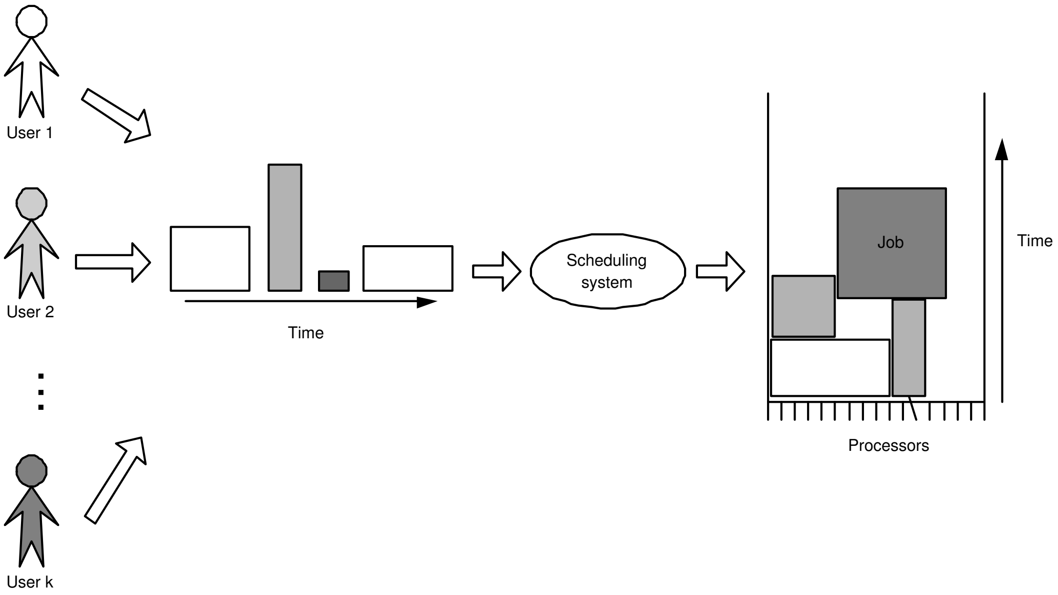

Figure 2.3: A medium number of users submit jobs to a parallel system.

2.2 Performance Evaluation a Using Workload Model

As we have pointed out, workload features need to be taken into consideration when designing a scheduling strategy. Although the scheduling problem is conceptually the same across different systems and parallel workloads, the feasibility and performance of possible solutions are very sensitive to the workloads [21, 23, 46]. There is no scheduling algorithm that is suitable for all scenarios. In other words, scheduling must be done with caution because solutions need to be carefully tailored according to the workload characteristics. Therefore, the evaluation of scheduling algorithms under different workload situations is an important step in designing a suitable scheduling system and setting appropriate parameters. Basically, there are several methods for performance evaluation of a scheduling system:

Theoretical Analysis

The theoretical analysis is usually a worst-case study. It tries to provide a theoretical bound for certain performance criteria. The worst-case study is only of limited help as typical workloads on production machines normally do not exhibit the specific structure that will really cause a bad case. In addition, the theoretical analysis is often very difficult to apply to many scheduling strategies due to its complexity. Therefore, it is seldom adopted in evaluating scheduling systems for real cases.

Simulation-based Analysis

In practice, simulation-based performance evaluation is often carried out. It simulates the working procedure of scheduling systems with software tools and then the performance can be obtained from the simulation results. Several simulation tools have been developed, for example, SimGrid [33].

One of the most important considerations about simulation is its input. This is, which kind of workloads should be used as an input for a simulator in order to simulate the performance under a real environment? Researchers have at their disposal two valid methods for conducting simulation: it (1) uses real workload traces gathered from real machines and carefully reconstruct for use in simulation testing, or (2) creates a model from real workload traces and use the model either for analysis or for simulation. Next, we will explain both of them in detail.

(1) Workload Traces for Simulations

A workload reflects a *real* test of a system: it records the job submissions precisely, with all their complexities even if they are unknown to the person performing the simulation.

The drawback is that a trace only reflects a specific usage of the machines: there are always doubts whether the results from a certain trace can be generalized to other situations. Moreover, the simulation of a scheduling system under different traces can be problematic. One reason is that if there are no enough jobs in a workload, the according simulation will not reflect the realistic performance of a scheduling system under heavy load. Since the traces were usually obtained from different resource configuration: it will be meaningless to conduct a simulation for a machine with 200 nodes using a trace obtained from a 100-node machine. It is because the trace will not contain the jobs whose nodes requirements are more than 100, which is obviously not true for the 200-node machine in practice. Another disadvantage is it is hard to change the characteristics of certain workload attribute(s), and even when it is applicable, it may be problematic. For example, it is difficult to increase the average runtime by adjusting the workload traces themselves. Increasing arriving rate by reducing the average inter-arrival time can be a problem, since the daily load cycle shrinks as well. If a model decomposes arriving rate and daily cycle, it will be feasible to adjust arriving rate as expected while keeping daily cycle unchanged.

(2) Workload Model for Simulations

In comparison with using traces, simulation using workload model has a number of advantages [17]:

- Model parameters can be adjusted stepwise, so that the investigation of individual settings can be performed while keeping other parameters constant. The stepwise parameter setting even allows the system designer to test how a system is sensitive to different parameters. It is also possible to select model parameters that are expected to match the specific workload at a given site.

- In many cases, only one experiment is not enough. Normally, more experiments are conducted in order to obtain certain confidential intervals. For example, a workload model can be applied several times with different seeds for random number generators.

- Finally, a workload model can lead to new designs of scheduling systems. A model is a generalization of a real workload and it is easy to know which parameters are correlated with each other because this information itself is part of the model.

With the deeper knowledge of workloads, the existing algorithms can be improved and even new methods can be derived. For instance, one can design a set of resource access policies that are parameterized by the settings of the workload model so that suitable resource policies are selected for different situations.

The key point of a workload model is its representativeness. That is, to which degree does the model represent the workload that the system will encounter? How are the crucial characteristics incorporated into the model? The answers depend not only on the methodology to build the model but also on the degree of details the model considers. In the next section, we will give a short overview on the existing methodologies for workload modeling in the domain of parallel computer.

2.3 Problems with Existing Approaches

Previous research focused on summarizing the overall features of the workload on a parallel computer [9, 36] as shown in Figure 2.4. Usually, the global characteristics of workload attributes are analyzed and certain methods are applied to summarize them.

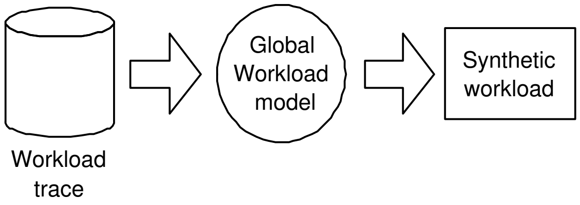

Figure 2.4: The global workload modeling structure

The summaries are a collection of distributions for various workload attributes (e.g., runtime, parallelism, I/O, memory). By sampling from the corresponding distributions, a synthetic workload is generated. The construction of such a workload model is done by fitting the global workload attributes to theoretical distributions. Normally, it is done by comparing the histogram observed in the data to the expected frequencies of the theoretical distribution. The modeling methods usually fall into three families [29]:

Moment-based: The kth moment of a sequence $x_1, x_2, \ldots, x_n$ of observations is defined by $m_k = \frac{1}{n}\sum x_i^k$. Important statistics derived from moments include: (a) the *mean*, which represents the "center of gravity" of a set of observations: $\overline{x} = \frac{1}{n}\sum x_i$; (b) the *standard deviation*, which gives an indication regarding the degree to which the observations are spread out around the mean: $s =$

$\sqrt{\frac{1}{n-1}\sum(x_i-\overline{x})^2}$; (c) the coefficient of variation, which is a normalized version of standard deviation: $cv = s/\overline{x}$.

Percentile-based: Percentiles are the values that appear in certain positions in the sorted sequence of observations, where the position is specified as a percentage of total sequence. Important statistics derived from percentiles include: (a) the *median*, which represents the center of the sequence: it is the 50th percentile, which means that half of the observations are smaller and half are larger; (b)*quartiles* (the 25, 50, and 75 percentiles) and deciles (percentiles that are multiples of 10). These give an indication of the shape of the distribution; (c) the Semi-InterQuartile Range (SIQR), which gives an indication of the spread around the median. It is defined as the average of distances from the median to the 25 percentile and to the 75 percentile.

Mode-based: The mode of a sequence of observations is the most common value observed. This statistic is obviously necessary when the values are not numerical, e.g., when they are user names. It is also useful for the distributions that have strong discrete components.

Here, we give several examples to explain how the classical methods are applied to model the workloads. Since the runtime and the parallelism of jobs are two of the most important attributes for many parallel systems [1, 47, 58], we focus on them in our study. The modeling of job arrival process is equally important and has been addressed by many papers, see [9, 35] for more detailed information about the job arriving process modeling.

As mentioned earlier, the job *runtime* is the duration that a job occupies a processor set. The runtime histogram of KTH is shown in Figure 2.5. It can be seen that runtime values usually spread from 1 to over 10^5 seconds. Such a distribution characteristic is called *heavy-tail* and can be formally defined as follows: a random variable X is a heavy-tailed distribution if

$$P[X > x] \sim cx^{-\alpha}, as\ x \to \infty, 0 < \alpha < 2$$

where c is a positive constant, and $\sim$ means that the ratio of the two sides tends to 1 for $x \to \infty$. This distribution has infinite variance, and if $\alpha \leq 1$ it has an infinite mean.

To model the heavy-tail runtime, Downey [15] proposed a multi-stage log-normal distribution. This method is based on the observation that the empirical distribution of runtime in log space was approximately linear. Jan et al. [30] proposed a more general model by using a Hyper-Erlang distribution for runtime. They used moment estimation to model the distribution parameters. Feitelson [22] argued that a moment estimation may suffer from several problems, including incorrect representation of the shape of the distribution and high sensitivity to sparse high value samples. Instead, Lublin & Feitelson [36] selected a Hyper-Gamma distribution. They calculated the parameters by Maximum Likelihood Estimation.

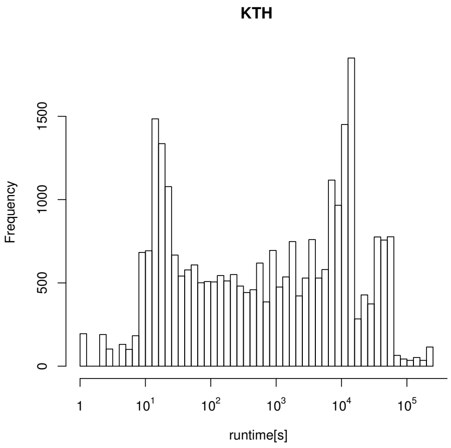

Figure 2.5: Histogram of job runtime

Another important aspect of workload modeling is the job *parallelism*, that is, the number of nodes or processors a job needs for execution. It has been found that the job parallelism in many available workloads displays two significant characteristics [15, 21]: (1) the power of 2 effect, as jobs tend to require power of 2 processor sets; (2) a high number of sequential jobs that require only one processor. These two features can be seen in Figure 2.6 clearly. It has been found empirically that these effects would significantly affect the evaluation of scheduling performance [35]. To describe these two features, a harmonic distribution is proposed in [21] which emphasize small parallelism and the other specific sizes like power of 2. Later, Lublin and Feitelson used job partitions to explicitly emphasize the power of 2 effects in the parallelism [36].

Besides the isolated modeling of each attribute, the correlations between different attributes were addressed as well. For instance, it has been found [30] that the runtime and the parallelism embody a certain positive correlation. That means the jobs with high parallelism tend to run longer than those with lower parallelism. Lo et al. [35] demonstrated that the neglecting of correct correlation between job size and runtime yields misleading results. Thus, Jann et al. [30] divided the parallelism into subranges and then created a separate model of the runtime for each range. Furthermore, Lublin

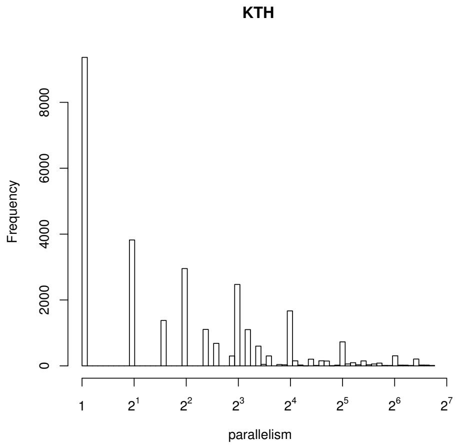

Figure 2.6: Histogram of job parallelism in the KTH workload

& Feitelson [36] considered the correlation according to a two-stage Hyper-Exponential distribution.

Although these models can provide the general description of a real workload, they have several serious drawbacks:

- Although static features can be characterized using probabilistic distributions, the temporal relation in job series is lost. A distribution model is based on the assumption of independent job submissions. As we mentioned earlier, due to the medium-sized user community, many user-level behaviors can still stay. Thus, such an assumption may not hold.

- The global level characterization does not provide an explicit explanation for user or user groups' behaviors and thus could not help to relate the global workload metrics with user groups. For example, a Hyper-Exponential distribution is used to describe the *heavy-tailed* runtime in parallel machine, but it can not interpret how this tail is generated; the high fraction of serial jobs is addressed by a harmonic distribution but it fails to explain where these serial jobs come from.

- User feedback on the quantity of service or system state is ignored. To take a simple example, some users may continuously submit jobs only if their previous jobs are finished. As a result, the submission of users is dependent on the scheduling results - a static model obviously does not address this point.

Based on the investigation of the existing models, we propose a novel workload model in the next section.

2.4 Proposal of a New Workload Model

According to analysis of the drawbacks of the existing models, we put forward to a new model. The structure of our model is shown in Figure 2.7. Instead of a general description of workload, our model considers several user groups, where distinctive submission behaviors are represented. For example, some users always submit the jobs with longer runtime; others tend to submit jobs requiring only one node. Our new model addresses the feedback of users. That is, users have adaptive behaviors - both the submission and profile of a job may be affected by quality of service or system state.

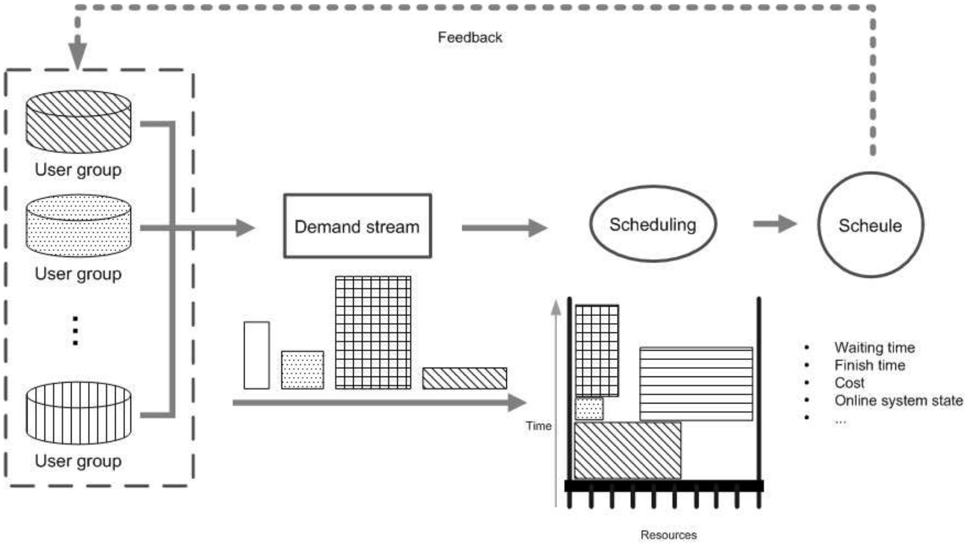

Figure 2.7: A novel workload model structure

In detail, the new model will be constructed according to the following steps:

Temporal Relations: First of all, we restrict ourselves on the global level of workload modeling. Previous models ignored the temporal relation. Hence it would be interesting to know whether there exist some temporal relation evidences. Here, several classical time series methods are considered, like ARIMA, Neural Network, Markov Chains.

Missing Information: Missing information is another problem to be dealt with when the global level modeling is considered. Due to different resource configurations and scheduling systems, certain attributes of the existing workloads are missing. Since there are not so many workloads available, we need to analyze these missing attributes and recover them so that the simulations based on different workloads can be performed. Here we consider the methods from classical statistics as options, e.g., analysis of variance, regressions.

User Groups: After analyzing the global workload features, we begin to the investigate user-level characteristics. Since the user community of parallel computers is medium-sized, it would be quite beneficial to associate the final workload with the user or user groups. With the user-level information it may be possible to explain those global features we have found and explore the effects of specific users on performance metrics. To identify user groups, the clustering methods from the data mining community can be applied, e.g., k-means, model-based clustering.

Additional Influencing Factors: Next, we investigate the possible factors that affect user submissions. Because of the lack of explicit influential factors, we need to derive certain implicit variables and try to identify feedback evidences. To this end, several techniques are tried to determine and measure the variable relations, like correlation and factor analysis.

Before we are turning to detailed explanation of our methods in the following chapters, we introduce the data set and software tools used in this work.

Workload Traces

The workload traces used in our work are from Standard Workload Archive [51]. They were collected from a variety of machines at several national labs and supercomputer sites in the United States and Europe. The type of workloads at these sites consisted of various scientific applications ranging from numerical aerodynamic simulations to elementary particle physics. Trace data were collected through a batch scheduling agent such as the Network Queuing System, LoadLeveler, PBS, or EASY [32]. Here, we briefly summarize the machine architecture, user environment, and scheduling policies of each workload:

KTH IBM SP-2: The Swedish Royal Institute of Technology IBM SP-2 machine with 100 nodes connected by a high performance switch. The trace came from June 1996 to May 1997 with scheduling managed by IBM's LoadLeveler.

CTC IBM SP-2: The Cornell Theory Center IBM SP-2 machine. The trace came from September 1996 to August 1997 with scheduling managed by IBM's LoadLeveler.

LANL CM-5 This log contains two years worth of accounting records produced by the DJM software running on the 1024-node CM-5 at Los Alamos National Lab (LANL). The trace came from periods: October 1994 to September 1996.

SDSC IBM SP-2: The San Diego Supercomputer Center houses a 128 node IBM SP-2 machine. This trace was taken from May 1998 to April 2000.

SDSC Intel Paragon 95: The San Diego Supercomputer Center houses a 416 node Paragon machine. The scheduling policies were implemented through the Network Queuing System (NQS). The trace was taken from December 1994 to December 1995.

SDSC Intel Paragon 96: The resource configuration was the same as for SDSC Intel Paragon 95. This trace was taken from from December 1995 to December 1996.

In Table 2.1 these traces are summarized for later references.

Identifier	CTC	KTH	LANL	SDSC SP2	SDSC 95	SDSC 96
Machine	SP2	SP2	CM-5	SP2	SP2	SP2
Period	06/26/96 05/31/97	09/23/96 08/29/97	10/04/94 09/24/96	04/28/98 04/30/00	12/29/94 12/30/95	12/27/95 12/31/96
Processors	430	100	1024	128	416	416
Jobs	79302	28490	201378	67667	76872	38719
Users	679	214	213	428	97	59

Table 2.1: Used workloads from the SWF Archive

Software Tool

The tool used in our work is R [5], which is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

We select R as our basic tool because R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, . . .) techniques, and is highly extensible. It is especially useful for our task, which in many cases is a try-out task. With R we can directly try various methods for modeling task.

R is good at producing well-designed publication-quality plots, including mathematical symbols and formulae where needed. Therefore, we can visualize the job submissions and identify the important aspects of data.

In addition, R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. We can obtain it freely from Internet, read and change the source code according to our needs.

Chapter 3

Investigation of Temporal Relations

As we have mentioned, workload modeling play a vital role in designing and developing the scheduling system for parallel computers. Therefore, we proposed a novel workload model in the last chapter. Our model consists of several fundamental components to address different aspects of the modeling problem. From this chapter on, we shall describe each component in detail.

In this chapter, we will focus on the temporal analysis of workloads. Many workload models use probabilistic distributions, which are based on the assumption of independent sampling. Therefore, we will verify whether such an assumption still holds or not in a parallel computer environment. Actually, our evidences show that there are strong temporal relations in job submission series. A straightforward methodology to model temporal relations is time series analysis, e.g., ARIMA. However, due to the relations between the job parameters, a direct application of the classical methodologies is infeasible. Hence, we propose a new approach not only to address the temporal relation but also to consider the parameter correlations. The experimental results will be discussed at the end of this chapter.

3.1 Observations on Temporal Relations

A temporal relation is an inter-propositional relation that communicates the ordering in time of events or states. Several temporal phenomena in job submission series have already been found. One of them is *repeated submission* [21], namely, users do not submit one job once but several similar jobs in a short time frame. Since there are only a medium number of users submitting jobs, such duplicated submissions can still be distinguished in the overall job submissions. It can be seen from Table 3.1 that a large number of neighboring jobs share the same parallelism values. This continuous occurrence demonstrates that the assumption of independent sampling is not correct.

	Percentage (%)
CTC	48.2
KTH	37.5
LANL	46.8
SP2	57.2
SDSC95	49.3
SDSC96	45.6

Table 3.1: Percentage of neighboring jobs with the same parallelism values

In addition, we find that not all jobs are submitted with the same continuity. For a job series J in a workload, we extract the parallelism $u_j \in U$ for each job j. We examine the average continued occurrences of parallelism values in the sequence U. That is, the number of direct repetitions is considered for each parallelism value in U. Note that we consider all job submissions instead of the jobs submitted by the same users. As the existing workloads contain predominantly jobs with the power of 2 parallelism values, we restrict our examination on such jobs requiring 1 node, 2 nodes, 4 nodes, etc. In Figure 3.1 the average subsequent appearances of job parallelism in real workloads is shown. As a reference, the average number of occurrences is provided if a multinomial distribution model is used for modeling parallelism. The details of the application of multinomial distribution can be found in [14]. This strategy models each parameter independently according to the statistical occurrences in an original trace. It can be seen that the sequences of the same parallelism values occur significantly more often in a real workload than it would be in a distribution model. This indicates that a simple distribution model does not correctly represent such an effect. Furthermore, it can also be seen that the jobs with less parallelism have a lager average repeating than that of jobs requiring more nodes in the real traces. That is, the jobs with less parallelism have a higher probability to be repeatedly submitted.

Even if those continuous appearing elements in U are removed, sequential dependencies can still be found. To this end, only one element is kept for each sequence of the identical parallelism values. For example, an excerpt in a series of parallelism of 1, 1, 1, 2, 2, 5, 5, 5, 8, 16, 16, 16, 2, 2 is changed to 1, 2, 5, 8, 16, 2 after the removal of repeated items. Here, the jobs with parallelism values that are not the power of 2 are considered as well. Suppose the transformed parallelism sequence is U'. Next, we

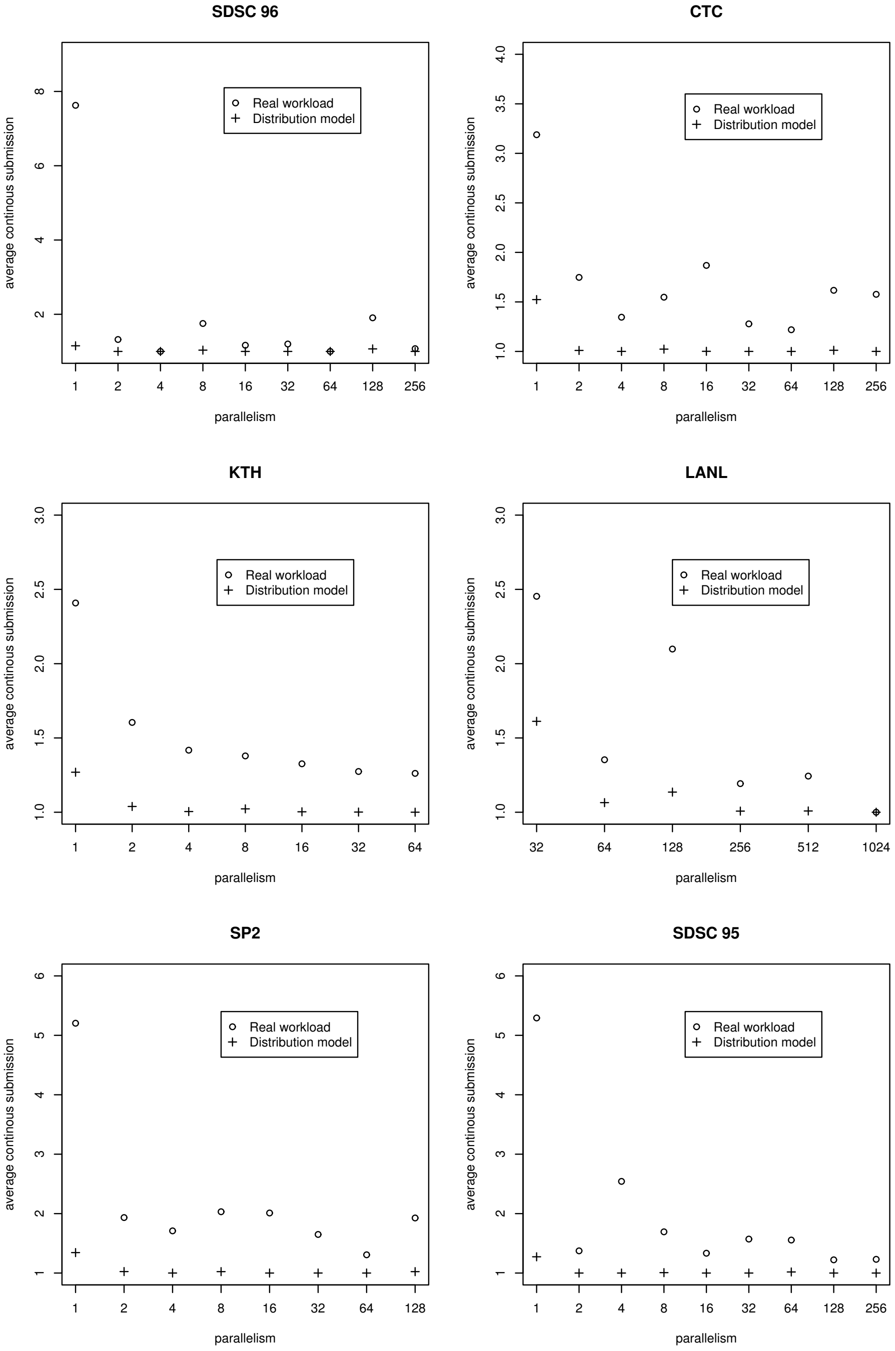

Figure 3.1: Continuous job submissions

transform U' to U'' by $U'' = \{2^{\lfloor log_2(u'_i) \rfloor} | u'_i \in U'\}$. That is, each parallelism value is rounded to the nearest lower power of 2. For each distinct parallelism value in U'', we calculate the average parallelism value requested by its successor in U'. The results are shown in Figure 3.2. The line in the figure is the overall average parallelism value as a reference. It can be seen that the successors of the jobs with a large node requirements also tend to request a large number of nodes for the most workloads. However, the behaviors for SDSC96 and SP2 are different from the others, the reason is not clear yet.

There is a temporal relation in the runtime series as well. To examine it, we group all the jobs by the integer part of the logarithm (based on 2) of their runtimes and for each group the average runtime of its successors has been calculated. The results are shown in Figure 3.3.

Such sequential dependencies may become very important for optimizing many scheduling algorithms, like e.g. Backfilling [24]. For instance, a scheduling algorithm can utilize probability information to predict future job arrivals. Such data can be included in heuristics about current job allocations. Therefore, a method to capture the sequential dependencies of workloads would be beneficial.

3.2 Modeling using Markov Chain Model

There are several classical methods to model a stochastic process. For example, Auto-Regressive Integrated Moving Average (ARIMA) time series models form a general class of linear models that are widely used in modeling and forecasting time series [7]. ARIMA has been successful applied in many applications where the continuous time systems are considered. However, since most parallelism values in the workloads are usually discrete, the ARIMA model is not suitable for our case. Another common approach is the use of Neural Networks to analyze and model sequential dependencies [11, 53]. But it is difficult to scale and extend such a model.

Therefore, a Markov chain model is chosen for modeling the described temporal patterns in Section 3.1. A Markov chain model has the important characteristic that the transition from one state to the next state depends on the previous state(s). To reduce the number of parameters in a model, we consider the application of first-order Markov chain. The first-order Markov chain can be described by a transition matrix. The element (i, j) within the matrix describes the probability to move from state i to state j if the system is in state i.

In our workload model, we use two Markov chains to represent the parallelism and the runtime respectively. If these two Markov chains are independent, they can not express the correlation between the parallelism and the runtime. Thus, more structures are required to reflect the correlation.

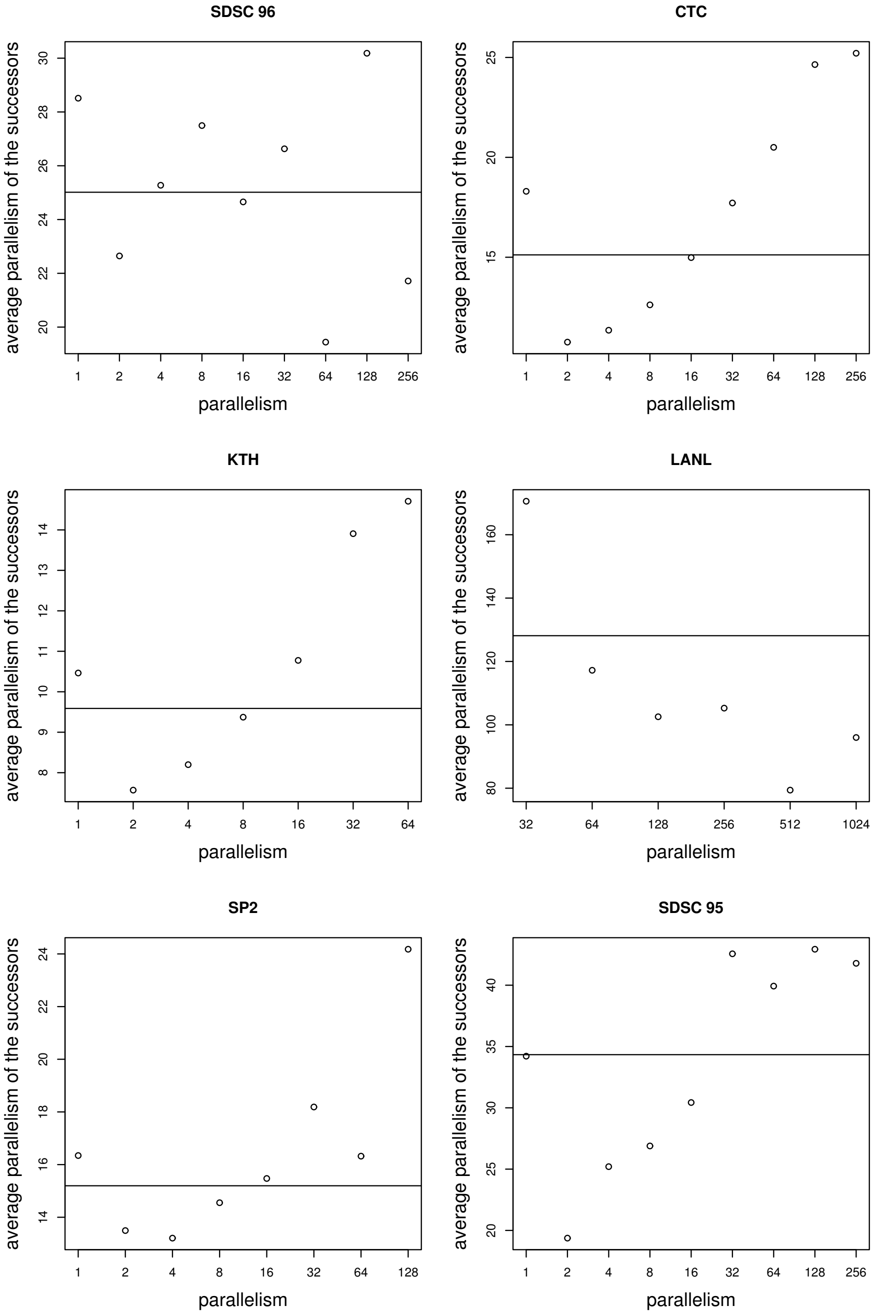

Figure 3.2: Temporal relation in the sequence of the parallelism

Similar requirements for correlating Markov chains also occur in other application areas [18, 39, 44]. For instance, advanced speech recognition systems use the so-called Hidden Markov Model (HMM) to represent not only phonemes, the smallest sound

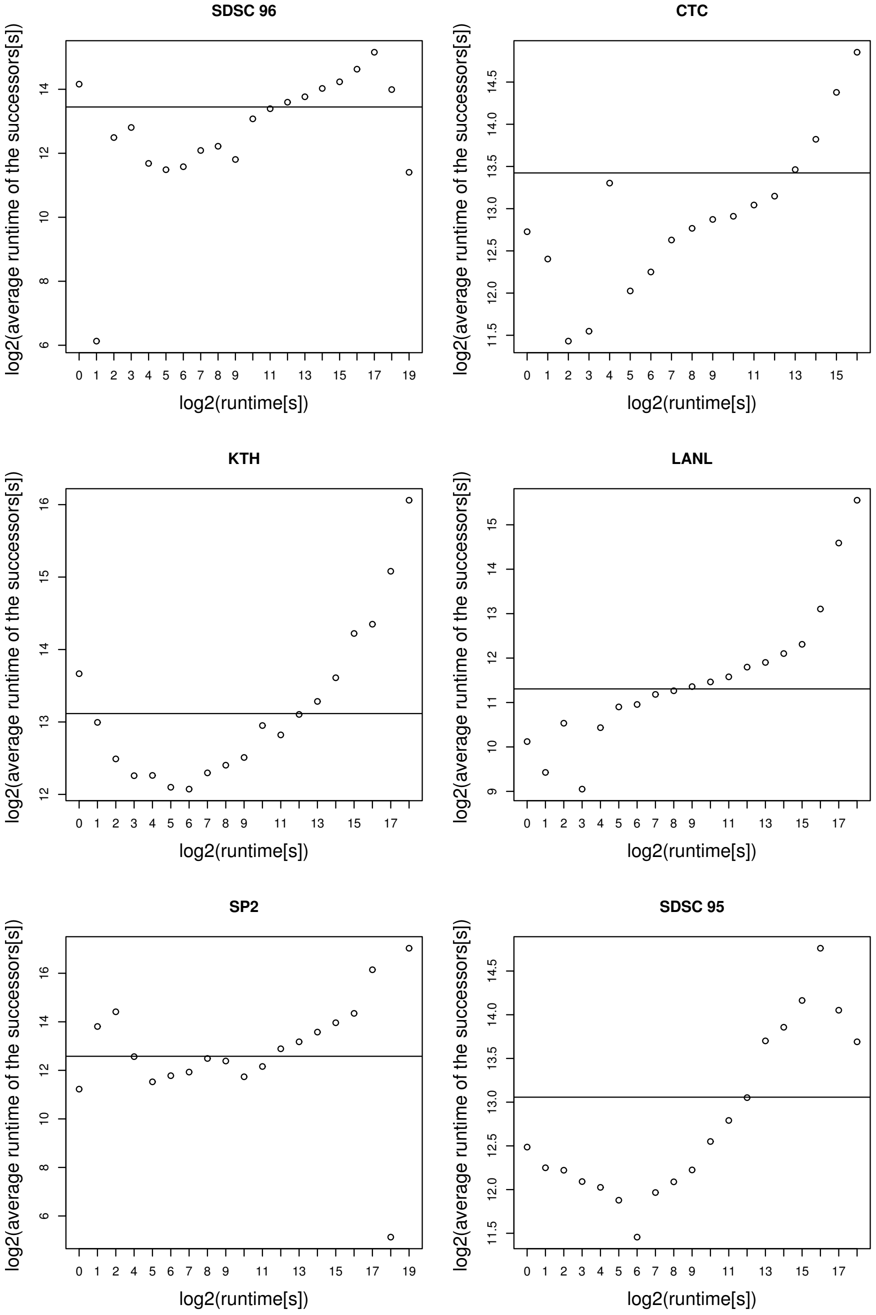

Figure 3.3: Temporal relation in the sequence of the runtime

units of which words are composed, but also their combinations to words. The method to correlate different Markov models is called "embedded" HMM, in which each state in the model (super states) can represent another Markov model (embedded states).

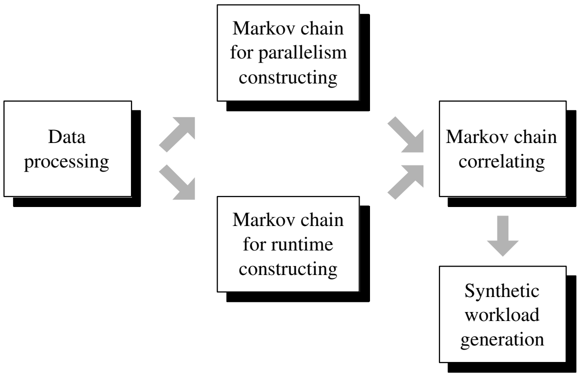
Data processing
Markov chain for parallelism constructing
Markov chain for runtime constructing
Markov chain correlating
Synthetic workload generation

Figure 3.4: Process of correlated Markov chains model

However, this method is not suitable to our problem, as it will dramatically increase the number of parameters. Consequently, the model becomes very hard to train. Hence, in the following we propose a new method to correlate two Markov chains without increasing the number of states.

Our model first builds two Markov chains independently, one for the parallelism and one for the runtime. Then the two chains are combined to address the correlation between parallelism and runtime. In Figure 3.4 the steps to build the correlated Markov chains model are shown.

3.2.1 Markov Chain Construction

Our method starts with constructing two independent Markov chains. Here, we explain how a Markov chain model for the parallelism is built, a similar method can be applied for the construction of the Markov chain for the runtime.

One of the key issues during the construction of a Markov chain is the identification of relevant states in the series. In our case, if all distinct values in the traces are specified as different states, the Markov chain would have a prohibitively high dimension transformation matrix. To this end, a small set of states are specified for the Markov chain for the parallelism.

Assume a sequence of n jobs where the series of the parallelism is described by the sequence $T = \{t_1, t_2, \cdots t_n\}$. Thus, the reduced sequence $S = \{s_1, s_2 \cdots s_n\}$ is constructed from T as follows:

$$s_i = 2^{\lfloor log_2 t_i \rfloor}, i \in [1, n] \tag{3.1}$$

Now, each *distinct* element in S can be considered as a separate state in the Markov chain. The set of states L of this Markov chain can be expressed as follows:

$$L = \{l_1, l_2, \cdots l_q | q \leq n; \forall i \in [1, q-1], j \in [2, q], i < j : l_i < l_j; \forall c \in [1, q], l_c \in S\} \quad (3.2)$$

Using this transformation, the original sequence T is represented using S and L. In order to consider the state changes in the original workload, we use U to denote the according transformation path of S. In fact, the sequence U is the corresponding indices of the elements in L corresponding to the job sequence. In Table 3.2 an example is given to illustrate the process of reducing the distinct parallelism values.

Index	1	2	3	4
$t_i \in T$	2	4	6	16
$s_i \in S$	2	4	4	16
$l_j \in L$	2	4	16	-
$u_i \in U$	1	2	2	3

Table 3.2: Example for deriving the states of the Markov chain for the parallelism

With U and L, the transition matrix E of the Markov chain for the parallelism can be calculated as: $e_{ij} = p_{ij}/p_i$, where e_{ij} and e_i are

$$p_i = |\{k | s_k = l_i, k = 1, \ldots, n-1\}| \text{ and} \quad (3.3)$$

$$p_{ij} = |\{k | s_k = l_i \wedge s_{k+1} = l_j, k = 1, \ldots, n-1\}| \; . \quad (3.4)$$

The transformation from T to S causes a loss of information about the precise parallelism, as they have been reduced to the power of 2 values. To record the information loss, a *quality ratio* c_j is defined for each state in the Markov chain. This ratio indicates how often the real values in the original group are exactly equal to the representing value in this state of the chain. More precisely, the quality ratio is calculated by:

$$c_j = \frac{|\{i | t_i = l_j, i = 1, \ldots, n\}|}{|\{i | s_i = l_j, i = 1, \ldots, n\}|}. \quad (3.5)$$

The definition of the quality ratios c_j, $j \in [1, q]$ is used to generate the final synthetical parallelism of a job. If the system is in state j, the corresponding value l_j is used as the system output with the probability of c_j. With the probability of $(1 - c_j)$ a uniform distribution between $[l_j, l_{j+1}]$ is used to create the final value for the parallelism.

The same method can be applied to model the runtime. This yields a second Markov chain. As a short summary, the dimensions of the two matrices for the considered workloads are presented in Table 3.3.

	Dimension of runtime chain	Dimension of parallelism chain
SDSC 96	19	9
CTC	17	9
KTH	18	8
LANL	18	6
SP2	19	8
SDSC95	19	9

Table 3.3: Dimensions of the Markov chains for the parallelism and the runtime

3.2.2 Combination of Two Markov Chains

As we have mentioned, the runtime and the parallelism have a weak positive correlation in all examined workloads (except CTC), that is, the jobs requiring more nodes have longer runtimes on average [23]. Such a correlation has an impact on the performance of the scheduling algorithms as shown in [21]. Therefore, this correlation should be reflected in our model as a key feature.

To this end, these two independent Markov chains for the parallelism and the runtime need be combined to incorporate the correlation. A common approach would be the mergence of these two Markov chains into a single Markov chain. However, this would yield a very high dimension chain based on all combinations of the states in the two original chains. Such a Markov chain is very difficult to analyze and could not be scaled for incorporating additional job parameters.

In our approach, we combine these two chains by adjusting the state of transformation of one chain depending on the state transformation of the other chain. Here, we describe how the new state in the Markov chain for the parallelism is adjusted according to the latest transition in the chain for the runtime. Using the transformation of the runtime chain to affect the transition of the parallelism chain is similar.

The idea is that since the runtime and the parallelism are correlated, the transformations of their corresponding Markov chains are related as well. For example, when the Markov chain for the runtime is in a state representing longer runtime, the state of the Markov chain for the parallelism would tend to move to a state requesting more nodes, when they are positive related, and vice versa. If the runtime changes dramatically in a chain, the parallelism would have a tendency to change correspondingly. As a result, the transformation of the states in the different Markov chains incorporates the correlation between their representing parameters.

In order to combine these two Markov chains, the correlations between their transformation paths cor_0 and their corresponding first-order difference sequences cor_1 are required.

Suppose the transformation path for the parallelism is denoted by $P = \{p_1, \ldots, p_n\}$, while $R = \{r_1, \ldots, r_n\}$ represents the runtime transformation path. Then the correlation cor_0 between the two transformation paths can be calculated as:

$$cor_0 = cor(P, R) = \frac{n \sum p_i r_i - \sum p_i \sum r_i}{\sqrt{n \sum p_i^2 - (\sum p_i)^2} \sqrt{n \sum r_i^2 - (\sum r_i)^2}} \tag{3.6}$$

The index 0 in cor_0 is used to specify that the transformation paths are used without further modifications. Furthermore, the first-order difference correlation cor_1 is used to denote how the changes of one transformation path affect the other. To define the first-order correlation precisely, some more variables have to be introduced. Therefore, two new sets are built which consist of the corresponding changes in the transformation paths:

$$\Delta P = \{\Delta p_i = p_{i+1} - p_i | i = 1, \ldots, n-1\} \tag{3.7}$$

$$\Delta R = \{\Delta r_i = r_{i+1} - r_i | i = 1, \ldots, n-1\}. \tag{3.8}$$

As shown in Section 3.1, the elements in a sequence often do not differ. It follows that the sequence of first-order difference includes many zero values. Since cor_1 is used to measure the change of one Markov chain affect the other, all elements are removed where either the parallelism or the runtime states do not change. This procedure leads to the new sequences $\Delta N'$ and $\Delta R'$, which are written as follows:

$$\Delta P' = \{\Delta p_i | \forall p_i \in \Delta P : \Delta p_i \neq 0 \wedge \Delta r_i \neq 0\} \tag{3.9}$$

$$\Delta R' = \{\Delta r_i | \forall r_i \in \Delta R : \Delta p_i \neq 0 \wedge \Delta r_i \neq 0\} \tag{3.10}$$

Now, the correlation cor_1 can be defined as: $cor_1 = cor(\Delta N', \Delta R')$. The actual values in our examinations for cor_0 and cor_1 are presented in Table 3.4.

With cor_0 and cor_1, the Markov chains for the parallelism and the runtime are combined. Here, three cases are considered: (a) if the Markov chain for the parallelism does not change, no adjustment is applied; (b) if the state changes only in the chain for the parallelism and not for the runtime, the state in the parallelism chain is adjusted based on the state in the runtime chain and cor_0; (c) if the states change in both

	cor_0	cor_1
SDSC 96	0.51	0.48
CTC	-0.08	-0.04
KTH	0.04	0.14
LANL	0.14	0.33
SP2	0.18	0.29
SDSC95	0.47	0.51

Table 3.4: Correlation cor_0 and cor_1 in examined workloads

chains, cor_1, the previous state of the parallelism chain and the last change in the runtime chain are used to adjust the state in the parallelism chain. Note, in order to reduce the number of parameters, we do not adjust the runtime chain.

Assume that the Markov chains for parallelism and the runtime have dimensions a and b respectively. Further assume that the synthetically generated transformation path of the parallelism is from state j to the state k, and the transformation for the runtime chain is from m to n.

The above mentioned procedure can be described in the following three rules in order to adjust the state in the Markov chain for the parallelism based on the Markov chain for the runtime:

1. If $j = k$, no transformation is applied. As the state in the Markov chain for parallelism is not changing ($j = k$), it is probably because of repeated submission effect of the parallelism series. Therefore, no adjustment in the Markov chain for parallelism is needed.

2. If $j \neq k$ and $m = n$, the destination state k is adjusted to $k = \lfloor n \cdot (a/b) \rfloor$ with the probability cor_0. This means that the resulting parallelism is changed with the probability cor_0 if the active state within the Markov chain for the parallelism changes while the state in the runtime chain stays constant. The factor a/b is used as a normalization between the two matrices. The value of n reflects the fact that the Markov chain of the runtime is used for the adjustment of the Markov chain for the parallelism. Here, a job with a longer runtime should also have a higher parallelism value, when the runtime and the parallelism are positive related.

3. If $j \neq k$ and $m \neq n$, the destination state k is changed to $k = \lfloor (n - m) \cdot (a/b) \cdot sign(cor_1) + j \rfloor$ with the probability $|cor_1|$. This rule is used in the situations where the states change in both of the Markov chains. Here, the incremental change is

used for the adjustment. The term $(n - m)$ describes the incremental change in the Markov chain for the runtime, where the $sign(cor_1)$ indicates the direction of the change. Again, the factor of a/b is used for the necessary normalization. As the first terms only describe the change, the originating state j is used as the basis. Similar to the step 2 the adjustment is only applied with the certain probability cor_1, which is calculated based on the incremental changes.

3.3 Experimental Results

To evaluate our approach, we have examined the workloads presented in Table 2.1. For all of these workloads, the corresponding Markov chains for the parallelism and the runtime have been created. Using the presented algorithm, the new synthetical workload traces have been created. The quality of the presented modeling method is measured by comparing the original with the newly generated traces using the following static and temporal criteria.

3.3.1 Static Comparison

To compare the static similarity, we use Kolmogorov-Smirnov (KS) test [34]. The KS test is used to decide if a sample comes from a population with a specific distribution or whether two sample sets are coming from the same distributions. The precondition of the test is that the observations are obtained independently, which is not met in our data. Since there are no other suitable methods to examine the static similarity of dependent data sets, we use KS test values as descriptive criteria. Then we take the results as a hint that the frequencies of states in the observed and simulated data are not too different. Furthermore, we calculate the difference of Squashed Area (SA) by

$$d_{\text{SA}} = \frac{\text{synthetic SA} - \text{original SA}}{\text{original SA}}. \tag{3.11}$$

It can be seen in Table 3.5 that the explained Markov chains models match well the original traces. The results for squashed area as well as for the KS test are quite acceptable. To take an example, Figure 3.5 shows the cumulative distribution curves of the original and synthetic runtime and parallelism in the KTH workload. In regard to the squashed area difference, it can be seen from the table that for the CTC workload the result shows an inappropriate deviation (about 38%). The squashed area or amount of total workload within a trace has a significant impact on scheduling performance [20]. However, information about this criteria is usually not provided for most workload models.

	KS test of parallelism	KS test of runtime	d_{SA}
SDSC96	0.08	0.06	8.3%
CTC	0.02	0.03	37.8%
KTH	0.04	0.03	15.4%
LANL	0.01	0.04	−1.1%
SP2	0.02	0.04	7.6%
SDSC95	0.09	0.02	−3.3%

Table 3.5: Static Comparison of the modeled and the original workloads

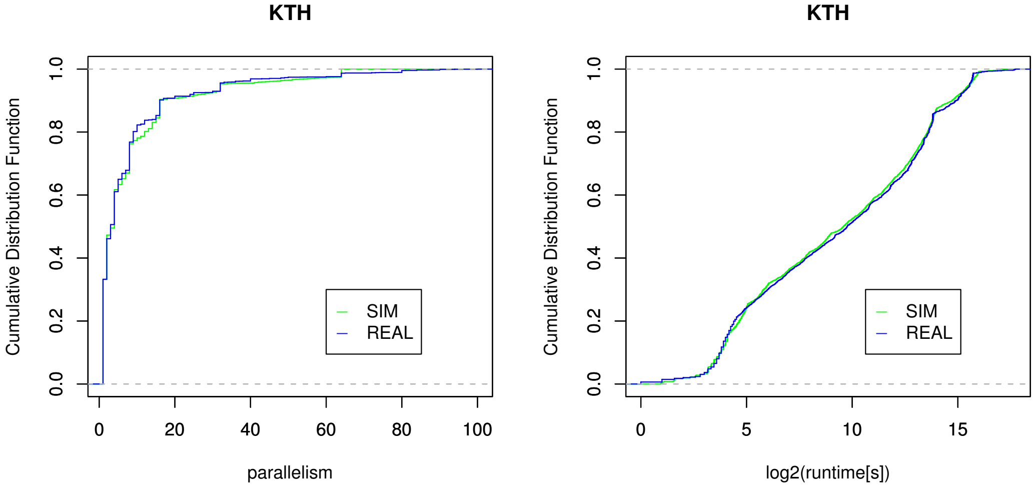

Figure 3.5: Comparison of modeled and original distributions of runtime and parallelism

We compare the presented model with the model by Lublin & Feitelson [36] in terms of correlation between the model and the original traces. That is, we compare the correlations from our model with that from real data and Lublin & Feitelson model. The Lublin & Feitelson model has been widely used for designing new scheduling algorithms or verify the existing scheduling systems, e.g., in [4, 52, 59]. It can be seen from Table 3.6 that our model is closer to the real correlation value than that in the Lublin/Feitelson model.

	Real Data	Markov chain	Lublin/Feitelson
		Model	Model
SDSC 95	0.28	0.14	0.11
SDSC 96	0.37	0.16	0.12
KTH	0.01	0.01	0.01
LANL	0.17	0.23	0.30

Table 3.6: Comparison of the correlation of the parallelism and the runtime from the Markov chain model and the Lublin/Feitelson model

3.3.2 Comparison of Temporal Relations

The autocorrelation ρ_1 has been used to examine the temporal dependency within each sequence. The results in Table 3.7 show that the Markov Chain Model correctly incorporates the temporal dependency since the ρ_1 from the synthetic data are *close* to the real data. As we see from the table, a Probability Distribution Model does not contain such a dependency.

	Parallelism series			Runtime series		
	Real data	MCM	PDM	Real data	MCM	PDM
SDSC 95	0.43	0.31	−0.01	0.28	0.16	0.01
SDSC 96	0.41	0.37	−0.01	0.17	0.20	0.02
KTH	0.29	0.29	0.01	0.29	0.30	−0.01
LANL	0.16	0.20	−0.01	0.18	0.19	−0.03

Table 3.7: Comparison of the autocorrelation ρ_1 of the parallelism and the runtime sequences from MCM, PDM, and the original workloads

3.4 Summary

In this chapter, a workload model based on Markov chain has been presented. This model does not only incorporate temporal dependencies but also keeps static similarity with original workloads.

The correlation between job parameters requires the combination of the different Markov chains. To this end, a novel approach of transforming the states in the different Markov chains has been proposed.

The quality of the modeling method has been evaluated with existing real workload traces. The presented workload model yields good results in comparison to the real traces. Here, the static characteristics as well as the temporal relation of our model are similar to those of the original workloads.

However, as to the overall modeling of workloads, there is another problem to be addressed - missing information in some workloads. This missing information prevents from a broad comparison of workloads under different resource configurations. Therefore, in the next chapter, we will discuss this problem.

Chapter 4

Analyzing Missing Information in Workloads

In this chapter, we concentrate on another important issue of the workload modeling for parallel computers - missing information in the workloads. Since the available workloads are obtained under various resource configurations and scheduling systems, the details of the workloads are not always uniform. It is not uncommon that some attributes of the workloads are not available. As a result, the simulations based on different workload situations can not be carried out. Therefore, in this chapter, a model is presented to recover the missing information. We take estimated runtime as an example to illustrate how a missing attribute is supplied in the workloads when it is absent. The quality of the modeled estimated runtime is evaluated by comparing different traces for which this information exists.

4.1 Missing Estimated Runtime

As we have shown in Table 2.1, there are only several workloads available which were recorded from real system installations. Due to the variety of different system configurations, it often happens that certain attributes are missing in some workloads. The absent information hinders the comprehensive comparison of workloads under the different circumstances and the broad verification of workload models.

One of missing attributes often encountered is estimated runtime. It differs from the real runtime in that the former is only the estimation of the real runtime by a user at a job's submission. Some scheduling systems require a user to provide such information when he or she submit a job. For example, Easy backfilling [60] uses this information to estimate the maximum delay of the queued jobs if a particular job is executed. When a job exceeds its estimated runtime, it is usually terminated by the scheduling system after a given time. As a results, the traces obtained from such systems contain the estimated runtime information, e.g., KTH, CTC. On the contrary, First Come First Service (FCFS) does not need the runtime estimation since it just assigns the resources to the earliest coming job. Hence, the traces obtained from a FCFS scheduling system do not have estimated runtime information. In Table 4.1 the missing estimated runtime is summarized for our examined workloads.

	SDSC 96	CTC	KTH	LANL	SDSC SP2	SDSC 95
Real runtime	Yes	Yes	Yes	Yes	Yes	Yes
Estimated runtime	No	Yes	Yes	Partially	Yes	No

Table 4.1: Summary of missing estimated runtime for the examined workloads

4.2 Analysis of Estimated Runtime in Complete Workloads

The missing estimated runtime has been noticed by Cirne & Berman in [9]. They modeled the estimated runtime and its accuracy independently. Based on these two parameters the real runtime is derived. The model requires the availability of the estimated and real runtime in the underlying workload traces to determine the corresponding model parameters. However, since the estimated runtime is missing in several existing workloads as shown in Table 4.1, this method can not be applied to deduce missing estimated runtime. Therefore, an appropriate approach is required. First of all, we shall analyze the estimated runtime from the complete workloads.

4.2.1 Estimation Accuracy

The Cirne/Berman model [9] assumed that a job can not run longer than its estimation. However, we found that it is not true for the complete traces as shown in Table 4.2. The table shows the *accuracy* of the estimated runtime which is defined in Equation 4.1. It also provides the information of the corresponding SA to indicate the resource consumption.

$$\text{accuracy} = \frac{\text{real runtime}}{\text{estimated runtime}} \tag{4.1}$$

It can be seen from the table that more than 10% of all jobs are exceeding their estimated runtime in the SP2, CTC and LANL workloads. Thereby, these jobs can not be neglected as they account for more than 20% of the total squashed area. However, we can also see that on average, with the exception of LANL, not many jobs exceed their runtimes by more than 10%. We neglect the jobs with an accuracy > 1.1 as they do not account for much amount of the total squashed area for the corresponding traces. For LANL, however, more than 20% of the total resource consumption is caused by those jobs with an accuracy > 1.1. Therefore, this particular workload trace is considered separately.

	accuracy > 1		accuracy > 1.1	
Traces	Percentage of jobs	Squashed area	Percentage of jobs	Squashed area
KTH	1.1%	1.8%	0.2%	1.1%
SP2	9.7%	26.1%	1.1%	1.1%
CTC	16.1%	20.3%	0.8%	1.2%
LANL	15.8%	30.5%	7.5%	22.7%

Table 4.2: Analysis of the jobs exceeding their real runtimes

We also found that the estimation accuracy depends on the real runtime. To this end, the jobs are grouped by the integer part of the logarithm (based on 2) of their runtimes. For each group, the average accuracy is calculated and shown in Figure 4.1. It can be seen that the accuracy increases with the real runtime. It may indicate that when users submit a longer job, they tend to give an accurate estimation.

4.2.2 User Estimations

Another point about the estimated runtime is that most users do not estimate an exact time span for the job execution time. Instead, they provide a general estimation, e.g., 1 minute, 10 minutes, 2 hours.

In Table 4.3 the most frequently required estimated runtimes from the workloads KTH, SDSC SP2 and CTC are summarized. It can be seen that the most jobs' estimated runtimes fit to these 20 groups (about 80 % of all jobs). This suggests either that the users tend to provide rounded estimates or that the estimated runtime is related to the configuration of available system queues with certain default values. The corresponding groups for the separate workloads are presented in Table 4.4 and Table 4.5.

4.3 Parameterized Distribution Model

In the following, we will model the estimated runtime based on the corresponding real runtime. As we have shown, the estimated runtime and the real runtime are correlated

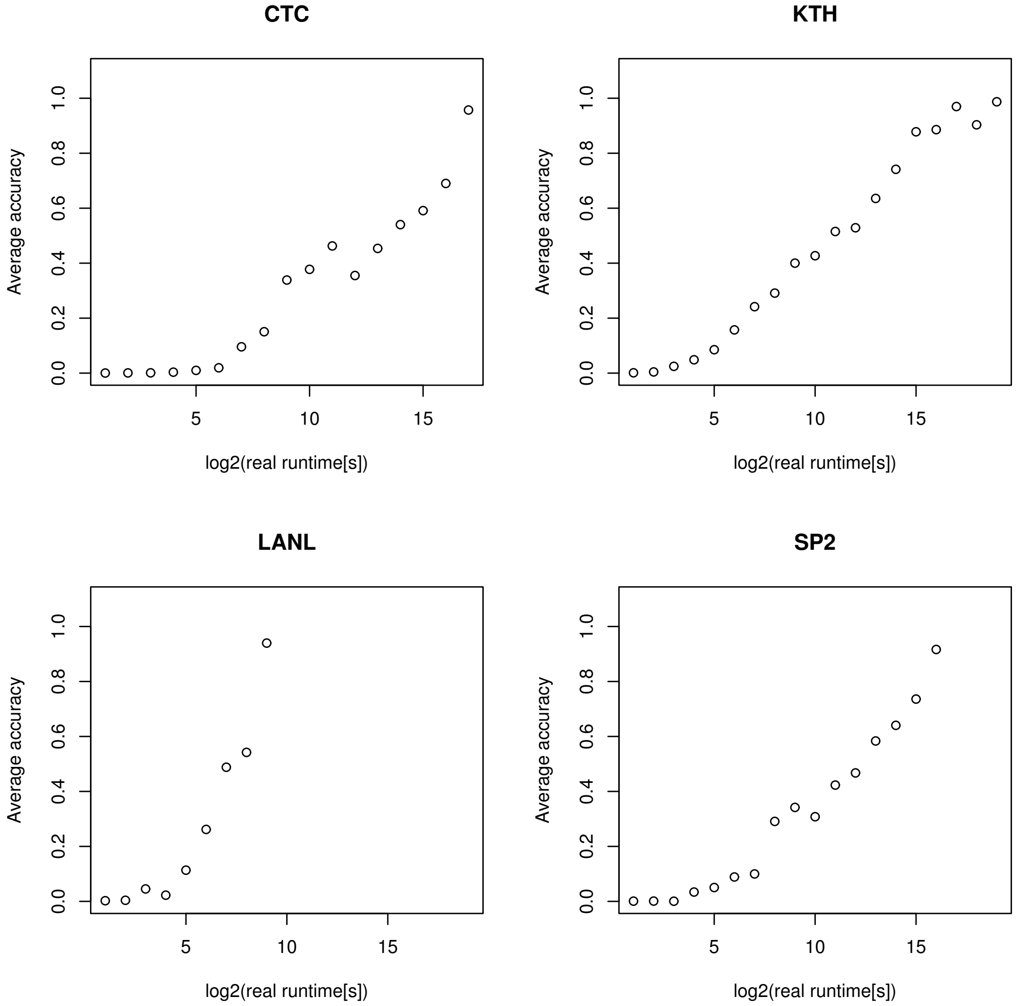

Figure 4.1: The relations of the accuracy to the real runtime. Here, only the jobs whose accuracies smaller than 1.1 are considered.

Group	Estimated runtime	Percentage of all jobs (%)	Group	Estimated runtime	Percentage of all jobs (%)
1	1 min	0.9	11	3 hours	3.9
2	5 mins	9.7	12	3.33 hours	0.8
3	10 mins	7.0	13	4 hours	5.0
4	15 mins	6.8	14	5 hours	1.2
5	20 mins	3.2	15	6 hours	4.8
6	30 mins	4.0	16	8 hours	1.9
7	1 hour	6.3	17	10 hours	2.3
8	1.5 hours	0.8	18	12 hours	2.5
9	2 hours	5.0	19	15 hours	1.1
10	2.5 hours	0.8	20	18 hours	14.4

Table 4.3: Summarized alignment of the estimated runtimes within the KTH, SDSC SP2, CTC workloads, total alignment: 82.7%

by the accuracy, therefore we model the accuracy first and then the estimated runtime can be calculated.

4.3.1 Model Selection

First of all, a basic model has to be selected to describe the accuracy. We select a Beta distribution [14] for the workloads of KTH, SP2, CTC, which gives a better fitting than the others, e.g., Gaussian. The general formula for the Beta distribution function is given in Equation 4.2 where p and q are the shape parameters, a and b denote the bounds of the distribution function. The Beta function itself is defined in Equation 4.3. On the contrary, for LANL a Gamma distribution is more suitable. The Gamma distribution is given in Equation 4.4, where Equation 4.5 describes Gamma function. Figure 4.2 shows the comparison of cumulative distribution functions (CDF) from the original data and the corresponding Beta distribution for KTH. It can be seen that a Beta distribution is capable of describing the accuracy.

	LANL		KTH	
Group	Estimated runtime	Percentage of all jobs (%)	Estimated runtime	Percentage of all jobs (%)
1	1 min	4.6	1 min	5.3
2	2 mins	0.9	2 mins	3.3
3	3 mins	1.0	3 mins	1.7
4	5 mins	23.3	5 mins	9.3
5	6 mins	0.3	10 mins	7.8
6	10 mins	5.6	15 mins	4.3
7	15 mins	3.8	20 mins	2.5
8	20 mins	2.0	30 mins	4.5
9	29 mins	0.9	1 hour	4.7
10	30 mins	4.7	1.67 hours	1.1
11	40 mins	0.9	2 hours	3.7
12	50 mins	1.1	2.5 hours	1.1
13	59 mins	0.8	3 hours	2.0
14	1 hour	25.8	3.33 hours	4.1
15	2 hours	0.7	3.83 hours	2.6
16	2.33 hours	0.6	4 hours	10.1
17	2.5 hours	0.9		
18	3 hours	16.5		
19	4 hours	0.3		
20	6 hours	1.0		
Total	96 %		74.9%	

Table 4.4: Alignment of the estimated runtimes within the LANL and KTH workloads

$$f(x) = \frac{(x-a)^{p-1}(b-x)^{q-1}}{B(p,q)(b-a)^{p+q-1}},\ a \leq x \leq b;\ p,q > 0 \tag{4.2}$$

$$B(\alpha,\beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt \tag{4.3}$$

$$f(x) = \frac{\left(\frac{x-\mu}{\beta}\right)^{\gamma-1} \cdot e^{\left(-\frac{x-\mu}{\beta}\right)}}{\beta\Gamma(\gamma)},\ x \geq \mu;\ \gamma,\beta > 0 \tag{4.4}$$

$$\Gamma(a) = \int_0^1 t^{a-1}e^{-t}dt \tag{4.5}$$

The independent modeling of the accuracy is not feasible for our problem, since the accuracy is related to the real runtime. Therefore, in the following section we will

	CTC		SDSC SP2	
Group	Estimated runtime	Percentage of all jobs (%)	Estimated runtime	Percentage of all jobs (%)
1	5 mins	8.6	5 mins	11.6
2	10 mins	6.2	10 mins	7.8
3	15 mins	10.4	12 mins	1.1
4	20 mins	1.9	15 mins	2.9
5	30 mins	3.4	20 mins	5.5
6	1 hour	4.1	30 mins	4.6
7	1.5 hours	0.8	45 mins	1.0
8	2 hours	5.3	1 hour	10.3
9	3 hours	4.8	2 hours	5.1
10	4 hours	2.1	2.5 hours	1.2
11	4.83 hours	0.6	3 hours	3.7
12	5 hours	1.1	4 hours	6.4
13	6 hours	8.5	5 hours	1.4
14	8 hours	1.5	6 hours	1.9
15	10 hours	1.7	7 hours	0.9
16	12 hours	2.2	8 hours	3.3
17	15 hours	1.4	10 hours	3.2
18	16 hours	1.0	12 hours	3.9
19	17 hours	0.6	15 hours	0.8
20	18 hours	23.1	18 hours	9.4
Total	89.1%		86.2%	

Table 4.5: Alignment of the estimated runtimes within the CTC and SDSC SP2 workloads

propose an integrated model for the estimated runtime.

4.3.2 Modeling Estimated Runtime

To reflect the relation between the accuracy and the real runtime we use a parameterized distribution model. That is, instead of using one accuracy model, a series of accuracy models are applied. These models are parameterized by the real runtime so that the relation can be represented.

First, in our model the jobs are grouped by their real runtimes. Then, for each of these groups, the parameter combinations of p and q of Beta distribution are identified in order to maintain a better similarity between the original and the synthetically generated workloads. Finally, the according accuracy distribution functions are derived from the combination of these parameters by a linear regression. In detail, the process

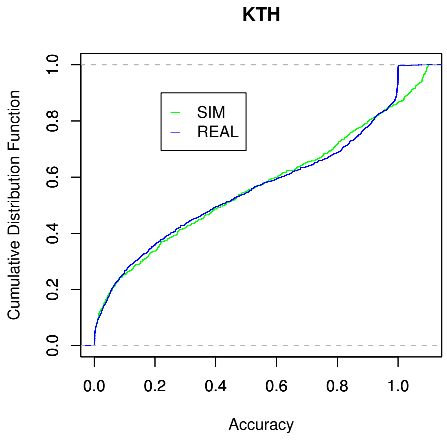

Figure 4.2: Comparison of modeled (SIM) and original (REAL) accuracies

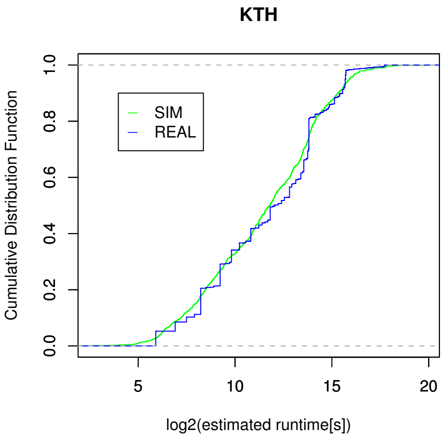

Figure 4.3: Comparison of modeled (SIM) and original (REAL) estimated runtime

works as follows:

1. In the first step all jobs are grouped by their real runtime. The jobs are grouped by calculating the integer of the logarithm (based on 2) of the real runtime. In our examples, up to 19 different groups for the different workloads are created. This allows the examination of the influence of the parameters p and q in a smaller subset of all jobs, as the real runtimes vary in a wide range. For each of these subsets a separate combination of the parameters p and q is generated. Suppose $\{t_1, \ldots, t_n\}$ are the real time values of n jobs. Then a group G_i is build as follows:

$$G_i = \{x | \lfloor log_2(t_x) \rfloor = r_i; x = 1, \ldots, n\},$$

   where k is the number of groups.

2. For each group G_i a combination of p and q can be found for which the Beta distribution resembles the accuracy in the group using moments estimates:

$$p = \bar{x}(\frac{\bar{x}(1-\bar{x})}{s^2})$$

   and

$$q = (1-\bar{x})(\frac{\bar{x}(1-\bar{x})}{s^2} - 1)$$

   , where $\bar{x}$ is the sample mean and s^2 is the sample variance. The results in Figure 4.4 present the relation between p, q and the real runtime for the KTH

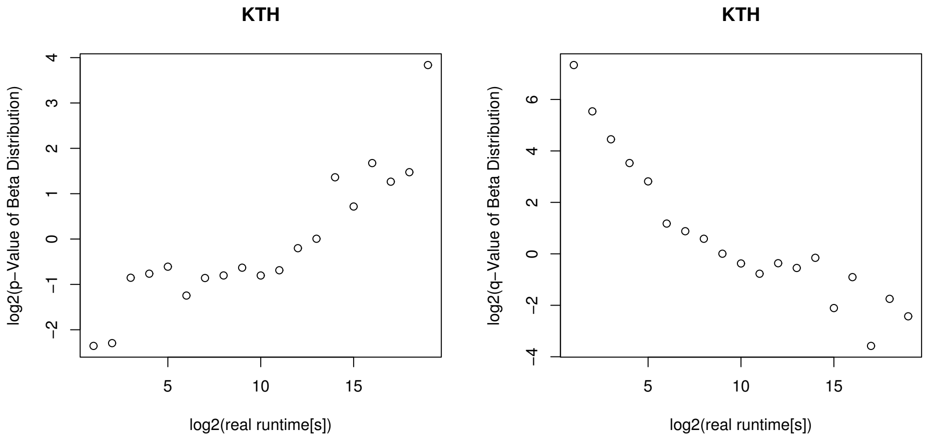

Figure 4.4: The relations between the real runtime and the parameters of the Beta distributions for KTH. The relations similar for the traces of SDSC SP2 and CTC

workload. As it can be seen that the parameters are not constant but p generally increases and q decreases with an increasing group number (created depending on their real runtimes). All other workloads have a similar behavior.

3. The parameters p and q are yet only described in a qualitative fashion. This step includes the determination of specific values for p and q depending on the real runtime. To this end we use a linear regression model for these two parameters. The linear model can be described as follows: $log_2(p) = log_2(T) \cdot a_1 + b_1$ and $log_2(log_2(q)) = log_2(T) \cdot a_2 + b_2$, here, T denotes the real runtime. For q we used the log_2 transformation twice as the results indicate a better fitting. For each group G_i a Beta distribution with specific p_i and q_i exists. So k pairs (r_i, p_i) and (r_i, q_i) can be used to derive the parameters (a_1, b_1) for p and (a_2, b_2) for q. The corresponding accuracy distribution can be created for given any the real runtimes using the resulting functions for p and q. The estimated runtime can be directly generated by using this accuracy distribution and the real runtime itself.

Figure 4.3 shows the curve of the cumulative distribution function using the described method to derive the estimated runtime. Here, the KTH workload is presented, but the other workloads show a similar behavior. The difference between the artificially generated and original estimated runtimes can be seen clearly in the figure. This difference is caused by the fact that most estimated runtimes have rounded values as we have shown before, e.g., 1 minute, 2 hours. Therefore, an alignment of the estimated runtimes can be used to improve the quality of our model.

To this end, the transformation method is extended by including an alignment process. From the available workloads we can see in Table 4.3 that between 70 and 95% of all jobs are aligned to some round values. Therefore, in our alignment process we decide that with a probability of 0.8 the estimated runtime is aligned to the nearest value according to the rule in Table 4.6.

As the synthetically generated estimated runtime might be much higher, the synthetically generated squashed area of the different workloads would be much higher than in the original traces. Existing scheduling strategies like Backfilling are sensitive to such a difference. To this end, the estimated runtime is bounded. Note, the runtime of all jobs is unchanged and so the real squashed area of all jobs of the different workloads is not affected. The bound of the estimated runtime is selected from the highest value of all jobs within the existing workloads. The maximum estimated runtimes of the examined workloads are presented in Table 4.7. Therefore, we have chosen an upper bound of 60 hours for the synthetically generated estimated runtimes.

Estimated runtime before alignment	aligned to multiples of
1 - 5 min	1 min
5 min - 60 min	5 min
1h - 4h	20 min
> 4h	1h

Table 4.6: Alignment of the modeled estimated runtime

	KTH	SP2	CTC	LANL
Maximum	60	48	18	8.33

Table 4.7: Maximum estimated runtime (hours) of the different traces

As mentioned in Section 4.2, the accuracy of the LANL workload has a wider range than the other workloads and so a Beta distribution is not suitable. Therefore, a modified procedure has been used for this workload. We have chosen a Gamma distribution [14] to model the accuracy for this workload. Here, the maximum estimated runtime for the alignment process is selected by 8.33 hours (Table 4.7). The process of parameterizing the Gamma distribution is similar to the method as described above; again, two parameters need to be extracted, which are γ and βinstead of p and q. The final selection of a reasonable estimated runtime is done by using the same alignment process as described for the other three workloads.

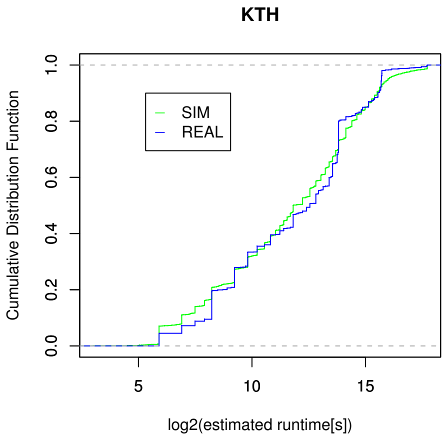

Figure 4.5: Comparison of the synthetic (SIM) real estimated runtimes (REAL)

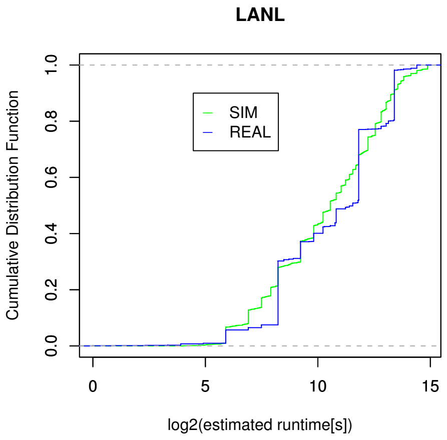

Figure 4.6: Comparison of the synthetic (SIM) and real estimated runtimes (REAL)

4.4 Experimental Results

4.4.1 Statistic Comparison

The empirical cumulative distribution functions of the synthetic and real estimated runtimes are plotted in Figure 4.5. The results of the KS tests and the comparisons of the estimated squashed areas of the workloads are shown in Table 4.8. Here, the difference of estimated SA is calculated by:

$$d_{\mathrm{SA}} = \frac{\text{synthetic estimated SA} - \text{original estimated SA}}{\text{original estimated SA}}.$$

	KTH	SP2	CTC
KS Test	0.06	0.14	0.15
d_{SA}	15.1%	-8.9%	34.1%

Table 4.8: Comparison of KS test results and difference of SA

The KS test shows that the synthetic and original estimated runtimes are similar. However, the results based on the comparison of the squashed areas are different for CTC. Here, the generated workload has an estimated squashed area 34% higher than that of the original workload. This is caused by our assumption of an upper bound for the estimated runtime of a job. As can be seen from the original workload trace of

CTC in Table 4.7, the upper bound is much lower than 60 hours. In order to increase the precision of the estimation a tighter upper bound can be applied for the estimated runtime in the alignment process.

4.4.2 Deriving a General Model

Next, we verify whether a general model can be derived to recover the estimated runtime for the traces without the information. Therefore, we train the model with KTH, SP2 and CTC and combinations of them. Finally, the synthetic estimated runtimes are compared with the original estimated runtime of the workloads, as shown in Tables 4.9. The KS test shows that the combination of KTH, SP2 and CTC is suitable to train the model for all 3 workload traces. In addition, the difference of estimated squashed area is given as well. It can be seen from the table (the first column of the SA comparison results) that the original value of KTH is smaller than the model trained by SP2, CTC or their combinations. This is due to the fact that the accuracy of KTH is better than that of CTC or SP2. Therefore, the overall accuracy of the estimated runtime is considerably worse than that of KTH. Based on the averaging effect by training with other workload traces, the lower modeled accuracy yields a higher estimated runtime. It follows that the synthetic squashed area is larger than its according original value.

	KS test results			SA comparison		
Training workloads	KTH	SP2	CTC	KTH	SP2	CTC
KTH	0.06	0.18	0.18	15.1%	−27.7%	−30.1%
SP2	0.10	0.14	0.15	39.7%	−9.4%	−9.8%
CTC	0.22	0.13	0.15	144.2%	40.3%	34.1%
KTH,SP2	0.14	0.11	0.13	48.6%	−15.9%	−12.3%
KTH,CTC	0.13	0.11	0.13	56.1%	−1.4%	−4.9%
SP2,CTC	0.17	0.10	0.11	95.5%	14.9%	7.1%
KTH,SP2,CTC	0.12	0.12	0.13	61.2%	−2.1%	−7.4%

Table 4.9: Comparison of synthetic estimated runtime and real estimated runtime

To use our model to recover the missing information, a decision must be made whether a workload resembles other workloads in order to train the model accordingly. The selection depends on the detailed knowledge of a workload itself.

For LANL, about 20% of the jobs have estimated runtime information, while it is missing for all other jobs. Therefore, we trained our model with those 20% of entries and used the model to recover the others. The cumulative distribution functions of synthetic and real estimated runtime are plotted in Figure 4.6. The results are acceptable with regard to a KS test of 0.13, and a difference of the squashed area of -7%.

4.5 Summary

In this chapter, we presented a model to recover an estimated job execution time for the workload traces without such information. This information is often used in scheduling strategies and therefore necessary for the evaluation process. The statistical criteria showed that the model produces acceptable results in comparison to the original traces. The evaluation results also showed that the parameterizations of the model varies for different workloads. The quality of the modeling depends on the similarity of the workload used for training and the workload scenario for which estimated runtimes are modeled. The presented method to characterize the estimated runtime is an example for recovering the missing workload attributes. Similarly, the method can be applied to other attributes.

Until now, we have restricted ourselves on analyzing the global features in the workloads. These overall features are useful for characterizing the final mixtures of job submissions. Whereas, in many application scenarios, a more detailed representation of the workloads is required. Therefore, in the next two chapters we go further to investigate user-level characteristics.

Chapter 5

User Group-based Analysis and Modeling

In the previous chapters, we have mainly concentrated on studying the overall characteristics of the workloads. Until now, the individual users and their correlations to final workloads are not considered yet. Since more job scheduling systems for single parallel installations as well as for Grid computing start to focus on the quality of service for specific users or user groups, the detailed knowledge of the individual users is necessary for developing user-oriented scheduling strategies.

From this chapter on, we begin to investigate the user-level behaviors. This chapter will mainly discuss the static characteristics of individual users. To this end, a new approach is given to cluster users into different groups according to their past submissions. Based on the analysis of the different user groups, a group-based model is proposed, called *MUGM* (Mixed User Group Model) , which maintains the features of the user groups. The comparison and analysis of different user groups will be given at the end of this chapter.

5.1 Analysis of User-level Submissions

When observing the individual submissions, one of the obvious characteristics in the examined workloads is the sparsity of users. That means, only a few users are responsible for thousands of jobs, while many other users just generate very few jobs. Table 5.1 shows the number of submissions from the users in the KTH workload: only a couple of users with more than 1000 job submissions. Actually, there are about 30,000 jobs from over 200 users. This effect is similar for all other workload traces.

Another aspect about the workload data is the heterogeneity of the job submission patterns among different users. In Figure 5.1 the average job parallelism values are plotted for the top 10 users with the most submissions: the heterogeneity can be seen clearly as some try to submit jobs with the lower parallelism (e.g. user IDs={91,93}), while some tend to make submissions with the higher parallelism (e.g. user IDs={18,67}). We have also examined the other workloads and found similar results.

# of Job Submission	# of Users (%)
$[1, 100[$	68.22
$[100, 500[$	18.22
$[500, 1000[$	12.15
$[1000, 2000[$	0.93
more than 2000	0.47

Table 5.1: Comparison of the numbers of users' submissions

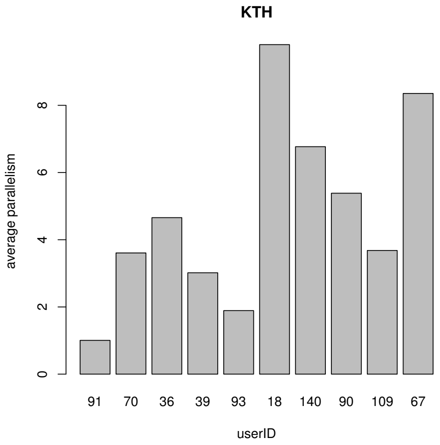

Figure 5.1: Comparison of the average job parallelism for individual users

The main challenge in the construction of a new model addressing the individual submission behaviors is the decision of a trade-off between two extremes. One extreme is the summarization of a general probability model for all job submissions. Thus, the user-level behaviors are not considered. As the other extreme, a specific model is created for each user based on his or her past transaction data, e.g., using hundreds of distributions for different users. Even if we would follow this approach, it suffers from two significant problems: there is not enough information for those users with only a few job submissions; even for those users with enough data to model, the number of parameters will be so large that the interpretability and scalability of the model are lost. As a consequence, we would like to get a mixture of user groups that summarizes similar user submission behaviors, while each of these groups has distinct features. Our proposed model allows to address the user submission behaviors, while it maintains the simplicity and the scalability. In the next section, we will give the details about our new model.

5.2 Clustering Users into Groups

Before we describe our MUGM model, some definitions are given in advance. We denote D as the set of n jobs by $D = \{d_1, \ldots, d_n\}$, where d_i represents the parameter set for job i, including e.g. the number of processors, the expected runtime, memory. This parameter set can easily be extended to contain additional job information. As previously mentioned, we currently focus only on the parallelism and runtime. Thus, we use d_i^p to represent the parallelism and d_i^r for the runtime of job i. The jobs are generated by J users, where user j generates job i: $d_i^u = j, j \in [1, J]$.

In our MUGM model a workload is analyzed to classify users into K user groups. Note that we do not assume that these K groups necessarily represent the *true* physical groups in the real environment. The membership of a user j is identified by $m(j) = k, k \in [1, K]$. The users in the same group are assumed to have a similar job submission behavior. Thus, the kth group $1 \leq k \leq K$ will represent a specific model for generating corresponding jobs.

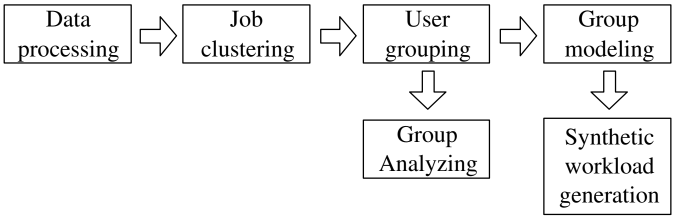

Figure 5.2: Process of the MUGM model

Figure 5.2 gives an overview on the construction of MUGM. We first find different job clusters or types using a cluster algorithm to partition jobs. In this step, the user origin is not considered. Instead, only common job types are identified by this clustering. Then each user is characterized by a feature vector, which describes the contribution of this user to each job cluster. Afterwards, we use these feature vectors to cluster user groups. In this way, the users are grouped by their similar contribution patterns to the previously identified common job clusters. Next, we analyze and model the submission characteristics of each user group. The combination of the group models is used to generate the synthetic workloads.

In the following, each step in the MUGM process is described in more detail.

5.2.1 Data Preprocessing

Parallelism and runtime values both cover wide ranges that are only bound by zero. Therefore, it is common practice to apply a logarithmic transformation for analyzing and modeling. Here, a log transformation based on 2 is used. That is, the parallelism and the runtime are transformed by $\log_2(d_i^p)$ and $\log_2(d_i^r)$. Zero values are neglected as they are very rare (less than 0.3%).

5.2.2 Job Clustering

Next, we cluster all jobs into several groups. The parallelism and the runtime are separately clustered in order to maintain sharper partition borders. That is, the different job clusters differ in the parallelism or the runtime. Each job is uniquely assigned to one of the clusters.

To cluster the parallelism we round all the job parallelism values $\log_2(D^p)$ to the $\lfloor \log_2(D^p) \rfloor$. Such a clustering method is based on the above mentioned observation of the power of 2 effect in parallelism. For the runtime we choose a clustering algorithm *CLARA* proposed in [31] because of its computational efficiency.

This algorithm is based on the Partition Around Medoids (*PAM*) method which is also presented in [31]. The *PAM* clustering procedure takes the unprocessed items as input and produces a set of cluster centers or so called "medoids". In the following, we briefly describe the general approach of the PAM method. Let $X = \{x_1, \ldots, x_T\}$ be the input element set of size T, and H be the number of clusters, and $M = (M_1, \ldots, M_H)$ denotes the list of to be mediods identified in X. The minimal distance of each element to the mediods can be calculated by

$$distance(x_t, M) = min_{h \in [1,H]}\{||x_t, M_h||\}.$$

Next, the PAM method selects the set of medoids M^* by minimizing the sum of the distances $f(M) = \sum_{t \in T} distance(x_t, M)$. An overview of the cluster algorithm PAM is given in Figure 5.3.

It has been pointed out in [31] that the complexity of a single iteration is $O(H \cdot (T - H)^2)$. Usually, $T \gg H$, therefore, it is computationally quite time consuming for large values of T. For instance, T in our evaluation equals the number of jobs which is larger than 20,000. The difference between the PAM and $CLARA$ algorithms is that the latter uses only a sampled subset $S \subset X$ before applying the same PAM method. This reduces the complexity to $O(H \cdot |S|^2 + H \cdot (T - H))$ for each iteration comparing to the $O(H \cdot (T - H)^2)$.

In our case, each element is a runtime value. We use the Euclidean distance between two logarithmic scaled runtimes, which is $(\log_2(d^r_{i'}) - \log_2(d^r_{i''}))^2$. To decide the number of clusters, we do not adopt classical methods like e.g. *silhouette,* Gap statistic [38, 54] because they caused a large number (more than 20) of small clusters which increase the complexity level for our MUGM model. In our analysis we tried several number of clusters, e.g. 2, 4, 6, 8, 10,..., 20. However, it turned out that the final modeling quality did not increase for more than 4 clusters. Note that it has to be verified if the model is applied to other workload traces.

Overall, all jobs have been partitioned into L clusters distinguished by the CLARA clustering of their log-transformed runtimes and the clustering of their log-transformed parallelism. In the next step, we group the users based on their contributions to the different job clusters.

5.2.3 User Grouping

We can characterize the submission of user j by a feature vector $\alpha_j = (\alpha_{j1}, \ldots, \alpha_{jL})$, where α_{jl} denotes the fractions of user j submissions belonging to job cluster $l, l \in [1, L]$. Obviously, there is $\sum_l \alpha_{jl} = 1, \forall j \in [1, J]$.

Then we can cluster all users into K groups by their feature vectors. The similarity of users is characterized by the distance of their feature vectors. Since different users have different number of job submissions, we weight the distance between the feature vectors of users by their corresponding number of job submissions. Here, the weight is used to prevent the appearances of tiny groups which have similar submissions but ignorable numbers of jobs. In more detail, the distance $d(j', j'')$ between user j' and j'' is defined by

$$d(j', j'') = ||\alpha_{j'} - \alpha_{j''}|| \cdot \frac{|W|}{n},$$

where $W = \{d^u_i | (d^u_i = j') \vee (d^u_i = j''); i = 1, \ldots, n\}$ and n is the total number of jobs. That is, we divide the number of jobs belonging to user j' and j'' by the number of all jobs, and then multiply the result by the distance between both feature vectors. With weighted distances between the feature vectors, the PAM clustering algorithm is again applied to partition the users into K groups. The determination of the actual number of groups K is given in the Section 5.4.

```
PROCEDURE $PAM\_clustering(X, H)$
Input: elements to be clustered,
      $X = \{x_1, \ldots, x_T\}$
  Input: $H$ the number of clusters
  Output: cluster medoids $M = (M_1, \ldots, M_H)$
  Output: cluster membership
      $g : X \rightarrow \{1, \ldots, H\}$
BEGIN
  Set $M$ to initial value,
  e.g. random selection from $X$
  FOREACH $t \in [1, T]$ LOOP
    $g(x_t) = arg \min_{h \in [1,H]} ||x_t, M_h||$
  END LOOP;
  DO LOOP
    FOREACH $h \in [1, H]$ LOOP
      recalculate the $M_h$ of
        each cluster $\{x_t | g(x_t) = h, t \in [1, T]\}$
    END LOOP;
    FOREACH $t \in [1, T]$ LOOP
      $g(x_t) = arg \min_{h \in [1,H]} ||x_t, M_h||$
    END LOOP;
  WHILE $M$ has not changed;
  END LOOP;
  RETURN $M, g$;
END PROCEDURE;
```

Figure 5.3: Algorithm for the *PAM* Clustering

5.2.4 Workload Modeling of Identified User Groups

After clustering the users into several groups, we characterize *all* jobs submitted from all the users in one group using statistical methods. There are several common methods to describe the data distribution. However, we found that after the users are grouped, the characteristics of jobs originated by each user group can not easily be described by a single distribution. Therefore, we use *model-based density estimation* to model the jobs from each group, which is described in more detail by Fraley and Raftery [27]. This model-based method assumes that the data is generated by a combination of distributions and determines the corresponding parameters for a set of Gaussian distributions.

We denote the estimated distribution function for a user group k as G_k. However, the G_k distribution function does not address the power of 2 effect for the parallelism. Hence, we extract the amount of power of 2 jobs f_k in the original workload of a user group k. That is,

$$f_k = \frac{|\{d_i^u | log_2(d_i^p) \in \mathbb{N} \wedge m(d_i^u) = k; i = 1, \ldots, n\}|}{|\{d_j^u | m(d_j^u) = k; j = 1, \ldots, n\}|}.$$

Additionally, the fraction of submission p_k from group k is calculated by

$$p_k = \frac{|D_k|}{n}$$

where $D_k = \{d_i^u | m(d_i^u) = k; i = 1, \ldots, n\}$. In a summary, the workload of the user group k can be represented by G_k, f_k, p_k. In the next section, we will discuss our method to generate the combined synthetic workload.

5.3 Generation of Synthetic Workloads

To create n synthetic jobs by the MUGM model, the following steps are applied:

1. For each user group k, we generate n_k jobs with $n_k = \lfloor n{\cdot}p_k \rfloor$ from G_k. We generate the synthetic parallelism $P_k = \{p_1, \ldots, p_{n_k}\}$ and runtime $R_k = \{r_1, \ldots, r_{n_k}\}$ by sampling from the distribution G_k. However, we also have to transform our previous log and round to the nearest integer value:

$$P'_k = \{p'_i | p'_i = \lfloor 2^{p_i} + 0.5 \rfloor; i = 1, \ldots, n_k\} \text{ and}$$

$$R'_k = \{r'_i | r'_i = \lfloor 2^{r_i} + 0.5 \rfloor; i = 1, \ldots, n_k\}.$$

2. To address the power of 2 effect, a fraction of the values in P'_k is rounded to the nearest power of 2 value. That is, with a probability of f_k the simulated value P'_k is modified.

3. The synthetic jobs from different user groups are combined. Particularly, we use probability p_k to pick a job from group k. According to this method we create the final n jobs.

In the next section, we discuss the evaluation of the MUGM method with experimental results.

5.4 Experimental Results

To evaluate our MUGM method we also used those 6 workloads from Standard Workload Archive as mentioned before. In the following, different user groups are analyzed and compared.

5.4.1 Analysis of Job Characteristic from User Groups

First, we examine the job characteristics of the resulting user group clusters. Here, the results for the KTH workload are shown primarily. However, the other workloads exhibited similar results. Figure 5.4, 5.5, 5.6 display the results for different numbers of user group clusters $K = \{2, 4, 6\}$. The user groups are ordered left-to-right and top-to-bottom in descending order by their combined amount of squashed area in each figure. The information in the header of each diagram shows the relative contributions of each user group to the squashed area SA, the total numbers of jobs and users. These figures give an idea of how much the parallel computer was utilized by one of the user groups.

The increase of K forces the creation of more user groups. For example, for $K = 2$, there are two user groups in Figure 5.4: the first group submits a lot of short jobs, requiring below 10 seconds; the other group causes more jobs with one node and longer runtime requirements. The parallelism is nearly not distinguished in this classification. For $K = 4$ more detailed user groups are found in Figure 5.5, in which combinations of runtime and parallelism are found. However, for $K = 6$ in Figure 5.6, it is noteworthy that some of the user groups cover only very few users with a small amount of workload. That is, for some of them SA contribution is even less than 1%. It can be deduced that these groups have a limited impact on the overall system performance. This has to be verified in the future research work.

Nevertheless, the results indicate that with 4 user groups the workloads could be distinctively covered. It is worthwhile to notice that this applied to almost all of the existing workloads as can be seen in Table 5.2. That is, there is only a limited number of distinctive features of user behaviors on real systems. It can be assumed that the additional user clustering only yields groups with minor contribution to the workload. Therefore, we focused on the creation of 4 user clusters.

For the KTH modeling with 4 user groups, the following characteristics of the user groups can be seen in Figure 5.5:

1. Users in Group 1 submit a lot of jobs with high parallelism. Many jobs require more than 1 node. Moreover, a large number of jobs run only for short time about 10 seconds. However, this group accounts for most resource consumption in terms of the SA. That is, over 40% of the total SA is caused by this group.

2. Users in Group 2 submit more jobs requiring relatively longer runtimes. Many of these jobs have a runtime more than 10^3 seconds. Some are even larger than 10^4 seconds. This group also creates some highly parallel jobs. But in comparison with Group 1, most node requirements in this group are less than 2^4.

3. It is interesting to note that the jobs from Group 3 are quite specific in terms of the runtime and the parallelism. Most jobs have a parallelism around 2^3 and a

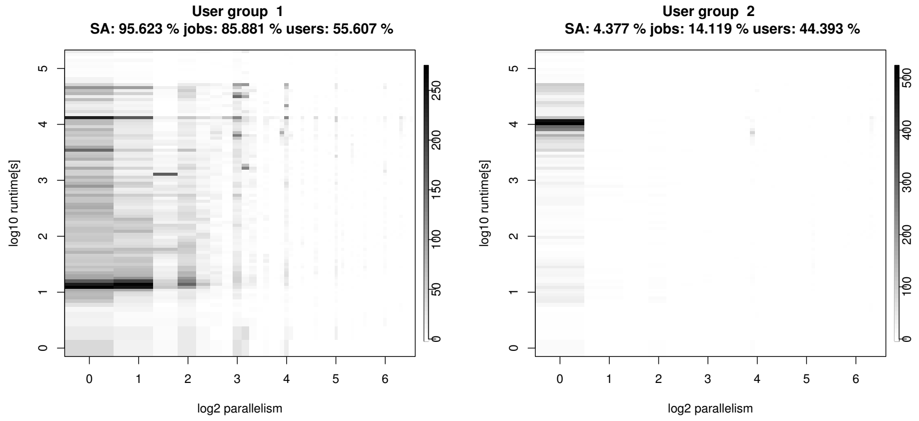

Figure 5.4: 2 user groups are clustered in KTH with the MUGM model.

runtime over 10^3 seconds. These users do not have jobs with short runtimes like Groups 1 and 2. The jobs account only for about 6% of the total job number but over 20% of the total SA. Another point is that only 3% users are in this group.

4. In Group 4, users concentrate on submitting sequential jobs, requiring only 1 node. The runtime is also distinctive: most of them run around 10^4 seconds. Over 40% of all users are in this group and they submit about 13% of the total jobs. It indicates that quite a lot of users use the machine primarily but infrequently for sequential jobs.

Note, that the other workloads do not exhibit the exactly same group characteristics, as can be seen exemplarily in Figure 5.7 for the LANL workload. However, as mentioned before about 4 user groups can be identified as well.

5.4.2 Statistical Comparison

The KS results are given in Table 5.3. It can be seen that the output of our MUGM method yields good results for most workload traces. That is, the KS value is below 0.10 in all cases and at 0.05 on average.

In Table 5.4 we present the correlation between the parallelism and the runtime for the synthetic and the original workload. As shown in the table the synthetic data from our MUGM model display a similar correlation as that from original data. It follows that our model can capture the correlation between the parallelism and the runtime.

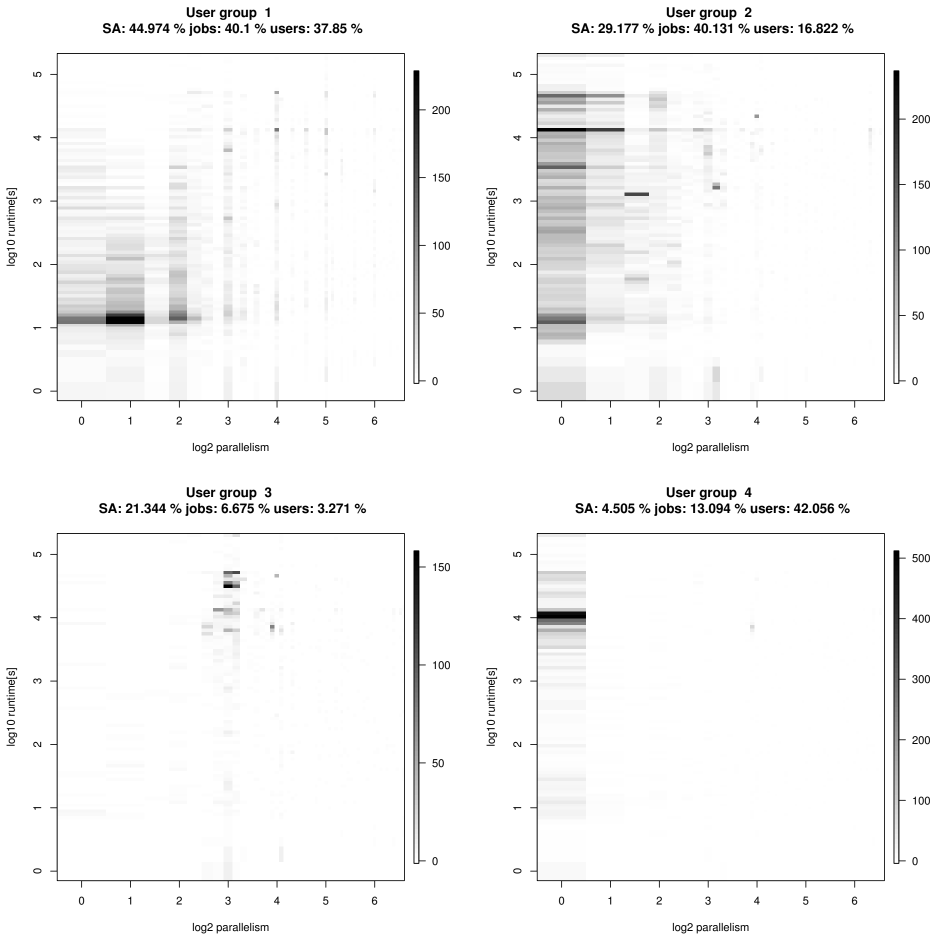

Figure 5.5: 4 user groups are clustered in KTH with the MUGM model.

5.5 Summary

In this chapter, we proposed a novel method MUGM for analyzing and modeling workload traces. The main advantage of this method is the consideration of individual user groups. Our MUGM method has been applied to several workloads from real installations. Here, it is interesting that the analysis of the workloads exhibited that only a few distinct user groups exist. This applied to all examined workloads.

The presented method allows the creation of the new synthetic workloads according to the original user group characteristics. This method can be used to evaluate new scheduling strategies. The job submission process has now a direct association with individual user groups. This information can be exploited for individualized quality criteria considered by scheduling strategies. Furthermore, additional workload parameters can be modeled in regard to individual scheduling objectives of these user groups. This applies especially to the Grid scheduling scenario in which the scheduling objective is not globally given for a specific computing system but depends on the user preferences.

Our MUGM model can explain the static behaviors of users but it does not reflect the feedbacks of users. Actually, users' submissions are not always dominated by their inner requests. They could also be affected by other dynamical factors. Therefore, in the next chapter, we will discuss users' dynamic feedback behaviors.

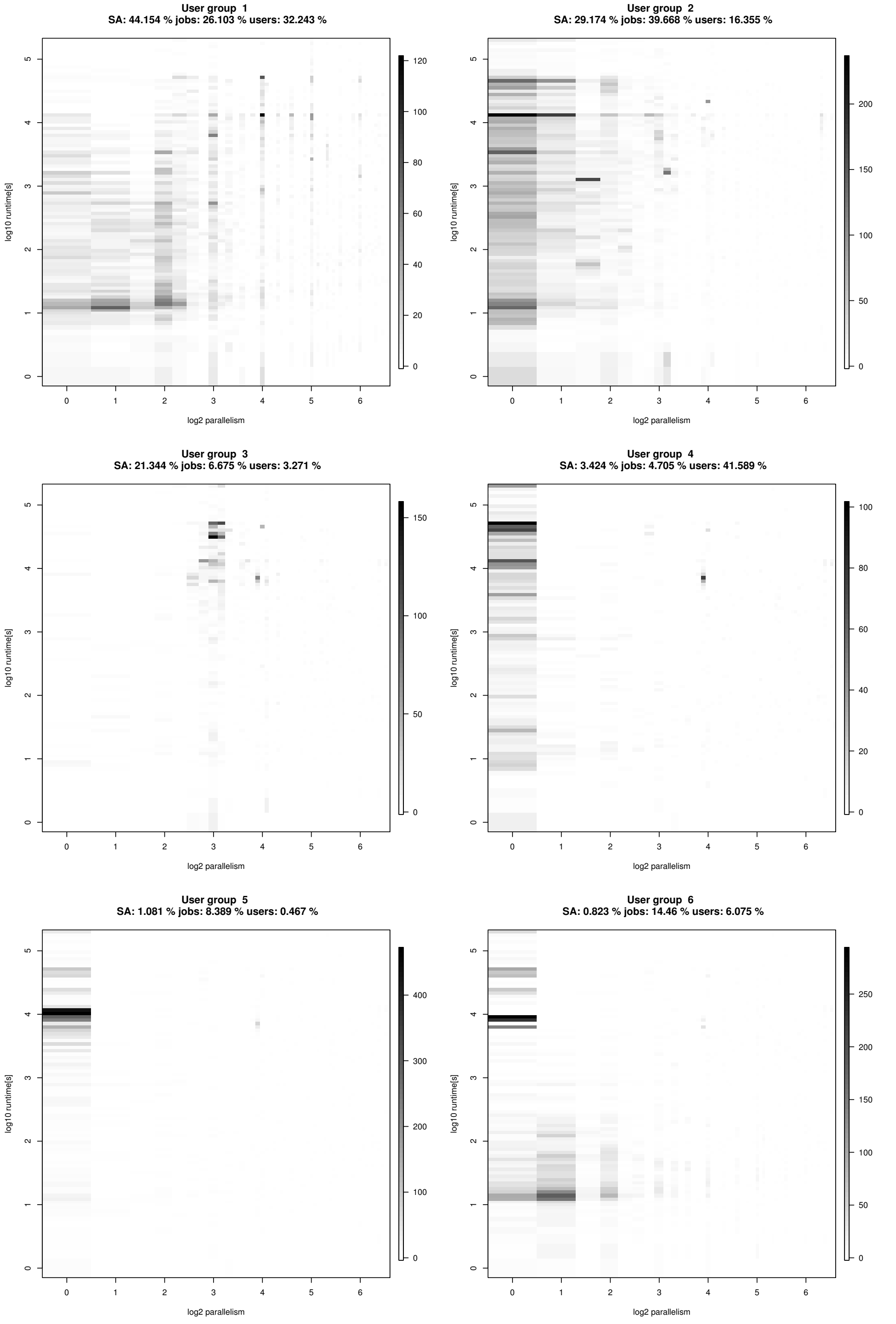

Figure 5.6: 6 user groups are clustered in KTH with the MUGM model.

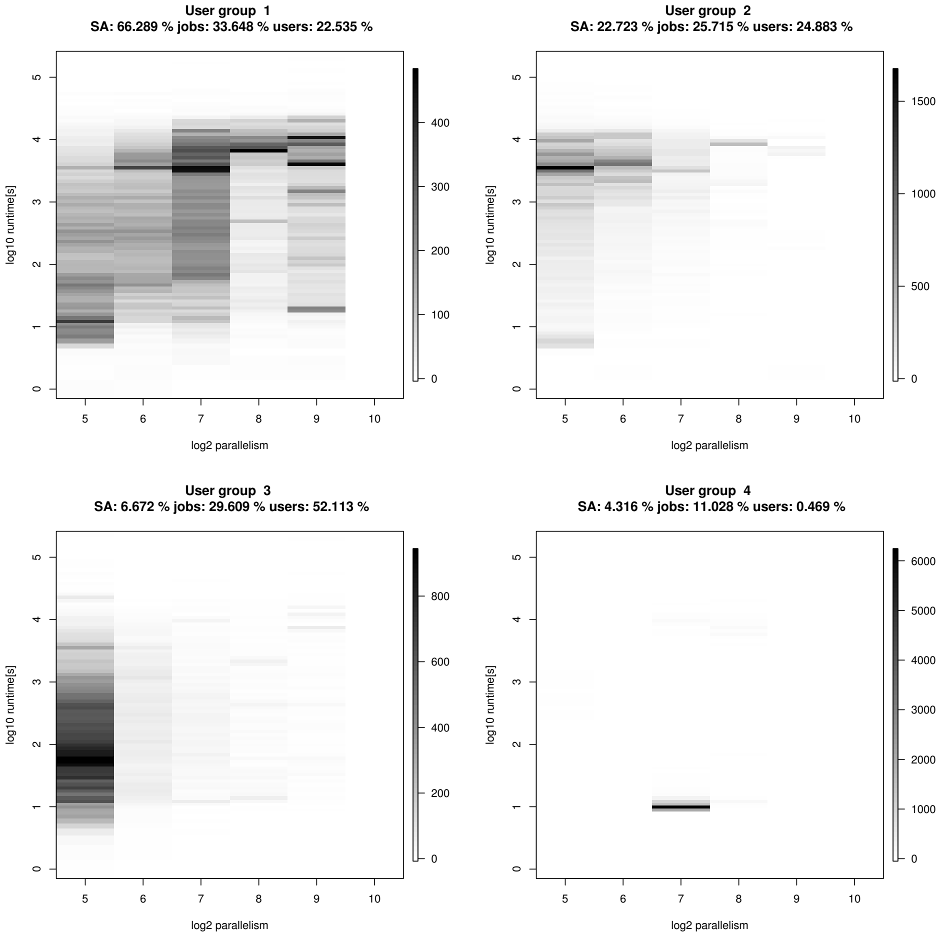

Figure 5.7: 4 user groups are clustered in LANL with the MUGM model.

Workload	K	User group information
SDSC96	2	(53.7, 80, 98.3), (46.3, 20, 1.7)
	4	(46.3, 20, 1.7), (22.1, 20.4, 72.9) (18.7, 40.0, 23.7), (13, 19.6, 1.7)
	6	(46.3, 20.0, 1.7), (20.6, 15.8, 71.2) (17.1, 6.6, 1.7), (13, 19.6, 1.7) (1.6, 33.5, 22.0), (1.5, 4.6, 1.7)
CTC	2	(77.7, 57.4, 30.0), (22.3, 42.6, 70.0)
	4	(75.9, 52.8, 26.7), (14.3, 24.1, 14.4) (9.0, 17.6, 30.8), (0.7, 5.5, 28.1)
	6	(49.5, 42.3, 17.4), (34.5, 15.2, 25.6) (11.1, 16.5, 10.9), (2.3, 6.8, 2.7) (2.1, 13.7, 16.1), (0.7, 5.5, 27.4)
KTH	2	(95.6, 85.9, 55.6), (4.4, 14.1, 44.4)
	4	(45.0, 40.1, 37.9), (29.2, 40.1, 16.8) (21.3, 6.7, 3.3), (4.5, 13.1, 42.1)
	6	(44.2, 26.1, 32.2), (29.2, 39.7, 16.4) (21.3, 6.7, 3.3), (3.4, 4.7, 41.6) (1.1, 8.4, 0.5), (0.8, 14.5, 6.1)
LANL	2	(95.7, 89, 99.5), (4.3, 11, 0.5)
	4	(66.3, 33.6, 22.5), (22.7, 25.7, 24.9) (6.7, 29.6, 52.1), (4.3, 11, 0.5)
	6	(53.2, 31.4, 22.1), (14.8, 19.9, 24.4) (13.1, 2.2, 0.5), (7.9, 5.9, 0.5) (6.7, 29.6, 52.1), (4.3, 11.0, 0.5)
SP2	2	(86.7, 71.6, 42.8), (13.3, 28.4, 57.2)
	4	(65.5, 53.2, 25.2), (16.1, 5.9, 3.7) (12, 17.7, 34.1), (6.3, 23.2, 36.9)
	6	(65.5, 53.2, 25.2), (16.1, 5.9, 3.7) (11.7, 15.7, 32.2), (5.7, 5.4, 1.6) (0.8, 2.6, 36.9), (0.2, 17.2, 0.2)
SDSC95	2	(99.7, 91.1, 99.0), (0.3, 8.9, 1.0)
	4	(76.5, 68.6, 96.9), (23.0, 5.1, 1.0) (0.3, 8.9, 1), (0.3, 17.3, 1)
	6	(68.9, 55.6, 94.8), (23.0, 5.1, 1.0) (7.5, 3.5, 1.0), (0.3, 8.9, 1.0) (0.3, 17.3, 1.0), (0.1, 9.5, 1.0)

Table 5.2: The Details of user groups (SA%, # of Jobs%, # of Users%) identified by the MUGM model

	Parallelism	Runtime
SDSC 96	0.06	0.03
CTC	0.04	0.06
KTH	0.05	0.07
LANL	0.04	0.06
SP2	0.03	0.05
SDSC 95	0.04	0.06

Table 5.3: KS test results (D_n) of the modeled and the original workloads

	Original	Synthetic
SDSC 96	0.37	0.41
CTC	-0.02	-0.01
KTH	0.01	0.00
LANL	0.17	0.14
SP2	0.15	0.09
SDSC95	0.28	0.24

Table 5.4: Comparison of the correlations between the modeled and the original workloads

Chapter 6

Examination of Implicit User Feedbacks

As we have mentioned, a deep understanding of users' behaviors has a very crucial implication for the scheduling systems of parallel computers or in Grid environments, e.g., resource reservation, load balancing. In the previous chapter, we have analyzed and characterized the static characteristics of users. The *dynamic* behaviors have not been discussed yet. Actually, users may change their job submissions according to online information, e.g., system states. In this chapter, we focus on investigating the responsive behaviors or *feedback* behaviors of the users. Due to the lack of direct information, we derive several implicit factors from the existing workloads and investigate how the users adjust their submissions by these factors; we also provide a method to identify and describe the details of feedbacks. At the end of this chapter, the results are compared among the different workloads.

6.1 Feedback Examination

The feedback analysis has a history in information retrieval that dates back over thirty years. It is widely used in short-term for modeling of users' immediate requirements and during long-term for modeling of users' persistent interests and preferences. Basically, there are two kinds of methods to extract the feedback information:

- *Explicit Inquiry*: This method requires that users explicitly give feedbacks by, for instance, specifying items, selecting from options, or answering questions about their interests. Such a method forces users to engage in additional activities beyond their normal submission tasks; the cost of inquiry can be high and the answers from users may be not correct. Hence, the effectiveness of this technique can be limited.

- *Implicit learning*: This method unobtrusively obtains the information about users by learning from their past submissions. The primary advantage to use implicit learning is the removal of the cost for extracting the feedback information. Moreover, the implicit measures can be combined with the explicit inquires to obtain a more accurate feedback representation.

Because of unobtrusive feedback obtaining, we consider to use the *implicit learning* method. There is a growing body of literature on implicit feedbacks, which is mainly in the fields of web navigation or network reservations [40, 42, 45]. In those fields there are explicit feedback variables available, e.g., preference rating and price information and then the feedbacks are analyzed directly.

In fact, the feedback behaviors may also exist in the field of parallel computers or Grid environments. For instance, some users who submit real-time interactive applications may be unable tolerate relatively long waiting time, whereas they may not care the waiting time when submitting a simulation program before returning home; some users may check the system load level before they submit a new job; some users make submissions only after all their previous applications are finished.

These feedbacks may be beneficial for machine owners, for example, to balance the loads and to improve the scheduling systems. New scheduling systems can be designed in regard to different user communities and their feedback tendencies. Therefore, in the following we will analyze users' feedback in the parallel computer environment.

6.2 Feedback Analysis

6.2.1 Implicit Feedback Factors

One problem about feedback analysis in our study is the missing of explicit feedback factors in the workloads. The available workloads themselves only recorded how jobs were submitted and scheduled, including job parallelism, runtime, submission time, etc. There is no explicit information about feedback factors, e.g., price, user satisfaction ratings.

Therefore, at the first step of the feedback analysis, the potential factors, which will influence users' submissions, need to be identified. As there are no explicit factors recorded in the workloads, we introduce two variables as implicit factors: the number of waiting jobs and the number of occupied nodes in the workloads, which are dynamically changed and can be regarded as the measurements of the system states. The concrete values can be easily calculated from the workloads and then we can investigate how these two variables affect job profiles. One can straightforward introduce more possible variables when more information is available. In short, we summarize the implicit feedback factors and job profile variables in Table 6.1.

After specifying the possible feedback factors we begin to explore the detailed feedback behaviors in the next section.

Name	Notation	Details
parallelism	P	the number of processors or nodes that a job requires
runtime	R	the duration that a job runs on the machine
the number of waiting jobs	W	the number of jobs which have been submitted but not been started yet at submit time
the number of occupied nodes	O	the number of nodes which are busy at a job's submit time

Table 6.1: Considered variables for feedback analysis

6.2.2 Problem Specification

To understand whether users' submissions are affected by certain factor(s) the following two questions need to be considered:

- Would the factor(s) affect a job's delivery? For example, would some users may stop delivering jobs when they notice a large number of waiting jobs?
- Would the factor(s) affect a job's profile, like job parallelism and runtime? For instance, would some users begin to submit the jobs requiring 10 nodes when they find there are over 10 nodes free in the system?

Due to lack of the standard references, it is infeasible to answer the first question. For example, if a user seldom delivers jobs at noon, it might result from a regular lunch at noon, or has a real feedback implication: the user finds many waiting jobs at noon and then stops his or her submissions. We also observed the histograms of W and Q by individual users and did not find obvious difference among users.

Whereas, it is possible to answer the second question, since the job profiles can be compared along different situations of influential factors. Therefore, we restrict ourselves on dealing with the second question: how would the given factor(s) affect the job profiles? That is, how would the number of waiting jobs and the number of occupied nodes affect the runtime and the parallelism of jobs for each user. In other words, the correlations between the feedback factors and the job attributes will be analyzed.

6.2.3 Method Selection

To investigate the correlation between variables, say X and Y, a straightforward method is the use of Pearson's *correlation coefficient*

$$\rho_{XY} = \frac{E((X - \bar{\mu}_X)(Y - \bar{\mu}_Y))}{\delta(X)\delta(Y)}$$

where $\bar{\mu}_X$ and $\bar{\mu}_Y$ are the means of X and Y respectively, $\delta(X)$ and $\delta(Y)$ are their standard deviations respectively.

However, the calculation of Pearson's *correlation coefficient* is very sensitive to the outliers. Very few outliers may deviate the results and it is difficult to identify the outliers. Another weak point is that when there are multiple variables available, it can only give pairwise correlation results and will not consider the combined effects.

Some methods can also be used to analyze the correlation between variables, e.g., non-linear regression, Neural network, factor analysis [56]. Since the goal of our study is to investigate and compare individual feedbacks so that the similar behaviors can be identified and explained, a model with many parameters is not suitable for our work.

As a result, we use a classical linear regression model to describe the relation between the feedback factors and job attributes for each user. The feedback effects may be much more complex than what a linear regression can describe. Whereas, the advantage of using linear models is that the feedbacks can be explained with few parameters. We can even identify user groups with similar feedback behaviors, and then arbitrate that certain behavior is worthy to consider or not in workload modeling. Next, we will explain the details of our method.

6.3 Methodology

6.3.1 Data Preprocessing

First, we preprocess the data in order to remove the noisy data. In our case, we need to find those irregular users and discard their submissions from the data set. We define the *irregular* users as those with less than 50 submissions. Since their submissions are rather few, we have not enough information to deduce their feedback behaviors.

Next, we should deal with the *skew* distribution of the data in the workloads. The data from skew distributions have the mode at a different value from the mean. To take one example, the histogram of the number of waiting jobs in KTH are plotted in

Figure 6.1. We can see the data are bounded at the left side (zero) and has a tail at the right side.

When we apply such data directly to the linear regression, the parameters of a linear model may be dominated by a small portion of the data. In our case, for example, the parameters may mainly depend on the jobs with longer runtime or higher parallelism. We can apply the logarithmic transformation. However, we find that the data are still unsymmetrical even after the transformation. Another way to deal with this problem is to discretize them. A common method, *Equal Frequency Intervals*, divides a continuous variable into k levels where (given m instances) each level contains m/k (possibly duplicated) adjacent values. In Table 6.2, we give an example to show how the number of waiting queue is discretized into 5 levels. In our work, we found that a large number of levels may cause noisy points and a small number of levels may be not enough for obtaining the parameters. As a result, we select $k = 10$ and use $0, \frac{1}{9}, \frac{2}{9}, \ldots, 1$ to represent the 10 different increasing levels. We apply such a discretization to the values of R, P, W and O at the global level. Using the transformed data, the corresponding linear models can be constructed. In the next section, we will begin to construct the linear models.

Level	1	2	3	4	5
Range	[0,8]	[9,13]	[14,17]	[18,23]	[24,+∞]

Table 6.2: Discretization Results with 5 levels for # of waiting jobs in KTH

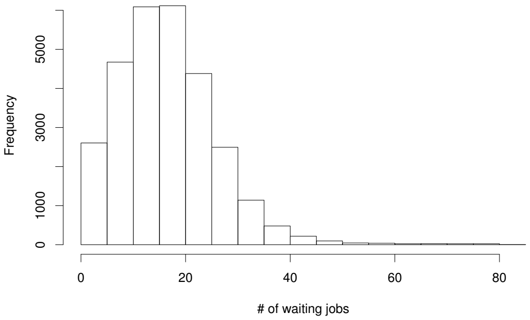

Figure 6.1: Histogram of the number of waiting jobs in KTH

6.3.2 Fitting Linear Regression Model

As mentioned previously, a linear regression model has been chosen to describe the variable correlations for each user. In detail, the linear model for user i takes the form:

$$R = b_i^{WR} \cdot W + b_i^{OR} \cdot O + \epsilon_i^R;$$
$$P = b_i^{WP} \cdot W + b_i^{OP} \cdot O + \epsilon_i^P;$$

where $B_i = \{b_i^{WR}, b_i^{OR}, b_i^{PR}, b_i^{OP}\}$ are the linear coefficients. Here, we can regard B_i as the descriptions of user i feedback behaviors, e.g., b_i^{WR} describes the correlation between the number of Waiting jobs and the Runtime. If $b_i^{WR} > 0$, it indicates that user i tends to submit longer jobs when the number of waiting jobs rises.

To obtain the parameter settings a stepwise regression procedure is used to decide whether b_i^{**} is 0 or not using the jobs submitted by user i. The stepwise procedure selects the subset of predictors optimizing on one of the following indicator statistics: Mallow's C_p [57], Akaike's Information Criteria (AIC) [2], R^2, or adjusted R^2 [14]. We select AIC since it tries to achieve a good compromise between the desire to explain as much variance in the predictor variables as possible (minimize bias) by including all relevant predictor variables, and to minimize the variance of the resulting estimates (minimize the standard error) by keeping the number of coefficients small. The stepwise regression procedure fits all possible combination of predictors and selects the model that results in the result which is close to optimal indicator statistic.

The AIC statistic, the likelihood version of the C_p statistic, is calculated as

$$AIC = \hat{\delta}^2(C_p + n)$$

and the C_p statistic is

$$C_p = p + \frac{(n-p)(s_p^2 - \hat{\delta}^2)}{\hat{\delta}^2}$$

where n is the number of observations; p is the number of parameters in the model, s_p is the mean square error and $\hat{\delta}^2$ is the estimate of error.

We apply the linear model for *each* user to optimize on AIC. That is, the jobs from the same users are used for training. Finally, the parameter settings of B_i are obtained for user i, with which the feedback details can be explained. The effectiveness of our linear model will be verified in the next section.

6.4 Experimental Results

In this section, we illustrate the experimental results using data from those workloads in Table 2.1. Here, LANL is not applied for verification, since its scheduling system used time-sharing strategies. It follows that a job can be started as soon as it is submitted and several jobs can share the same nodes in turn. Thus, the number of waiting jobs is always equal to 0 and the number of occupied nodes cannot be calculated from the workload. Therefore, the evaluations are obtained from the other 5 workloads.

The evaluations are based on the analysis and visualization of users' feedback behaviors. As an example, we give a detailed explanation of the results for the workload KTH.

6.4.1 Feedback Visualization

Using our linear model, the feedbacks of users can be identified directly. For instance, user j who tends to submit shorter runtime jobs when the number of waiting jobs increases can be distinguished by $b_j^{WR} < 0$. In this way, we obtain several *overlapped* user groups with similar feedback behaviors:

$$B_+^{WR}, B_-^{WR}, B_+^{OR}, B_-^{OR}, B_+^{WP}, B_-^{WP}, B_+^{OP}, B_-^{OP};$$

where J is the number of users and $B_+^{WR} = \{b_i^{WR} | b_i^{WR} > 0, i \in 1, \ldots, J\}$, $B_-^{WR} = \{b_i^{WR} | b_i^{WR} < 0, i \in 1, \ldots, J\}$, etc.

To give a straightforward illustration on discovered feedback behaviors, we visualize the job submissions from each group. Due to unbalanced submissions, some user groups may contribute only a small number of jobs comparing to the total jobs in a workload. Thus, not all the user groups are considered in our study. Here, only those groups with their total submissions over 20% of whole entries are displayed as the *evidence* of feedbacks. Note, the user groups are overlapped. That is, a user can belong to different groups at the same time. The results are shown in Figure 6.2, where the average runtime and the average parallelism are plotted under different levels of implicit feedback factors for each user group. Additionally, the global average value in the workload is given (horizontal line) as a reference.

Several interesting feedbacks have been identified: as shown in the top-left part of the figure, when considering the jobs from the users in B_-^{WR}, the average runtime decreases from about 4.5 hours to about 2.0 hours with the increasing of the number of waiting queue levels. These users may submit longer jobs when they find that few jobs are waiting for the machine. Correspondingly, from the top-right part of Figure 6.2 we can see that the runtime is affected by the number of occupied nodes as well: the average runtime rises when the number of occupied nodes increases.

In the bottom-left and bottom-right parts of the figure, the feedbacks on parallelism are displayed. It can be seen that for the users in B_+^{WP} average parallelism values grow when more jobs are waiting. On the contrary, the users in B_-^{WP} will slightly decrease the parallelism. It is also worth noting that the average parallelism of jobs from the users B_-^{OP} declines with the rise of the number of occupied nodes levels. This can be caused by the fact that some users tend to submit jobs with high parallelism when they observe that more nodes are free.

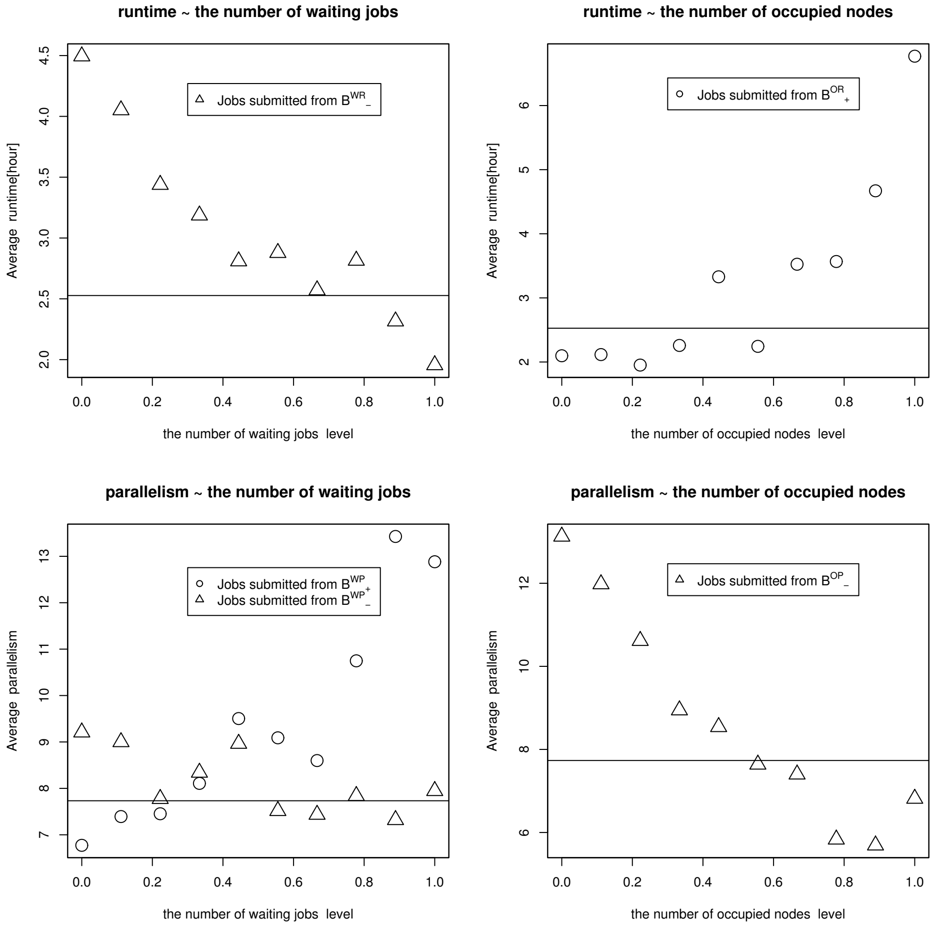

Figure 6.2: Feedback discoveries in KTH

Note, the other workloads do not exhibit the identical feedback evidences, as the user communities are different. However, some feedbacks are found as well. In Figure 6.3 the results for CTC are given as another example.

6.4.2 Exclusion from Other Factors

In order to ensure that the discovered feedbacks do not inherit from other factors, we need to exclude the effects of other possible factors.

We take the daily cycle as an example. Daily cycle is referred as the phenomena that more jobs arrive in the day while few jobs arrive at night. Therefore, the number of waiting jobs in the day tends to be larger than that at night. Users could submit a

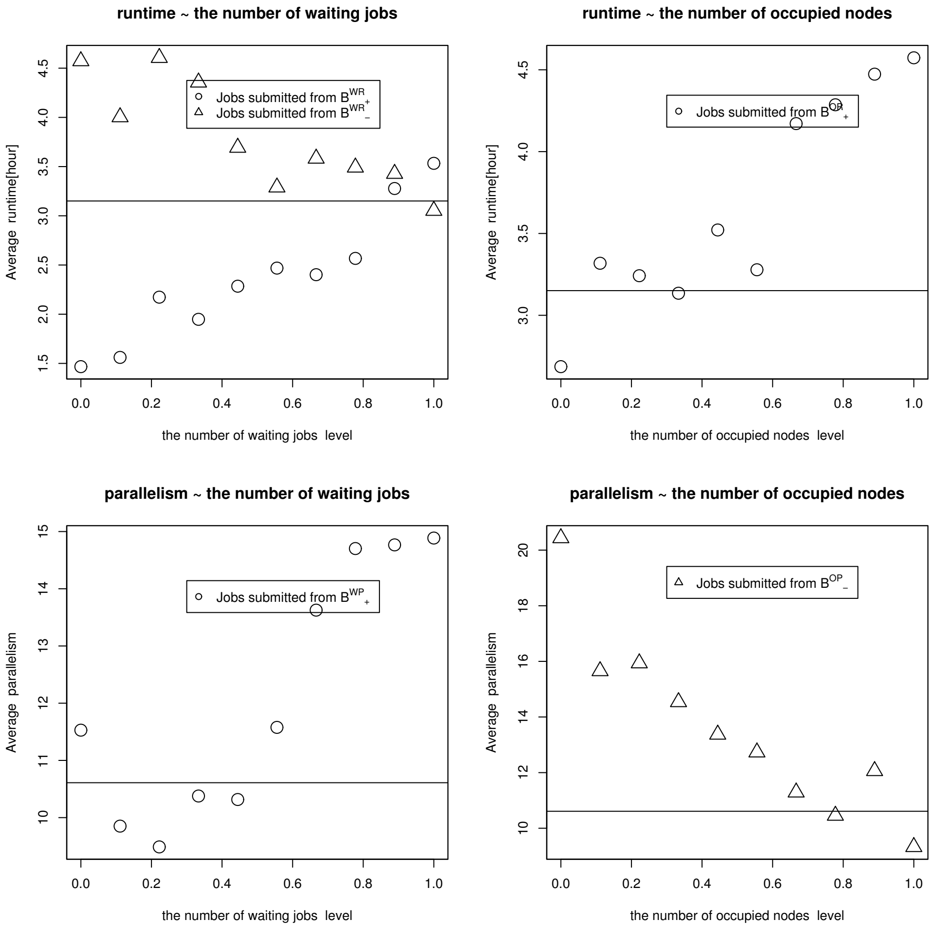

Figure 6.3: Feedback discoveries in CTC

certain kind of jobs at specific time span regardless of waiting jobs, e.g., user submits jobs requiring the longer runtime at night. As a result, the conclusion can not be made that users' submissions are affected by the number of waiting jobs.

To remove daily cycle influence, we focus ourselves on small time frames. For instance, we take the time frame from 1 pm to 4 pm and display the jobs appearing in that time frame in Figure 6.4. The feedbacks still exist comparing to Figure 6.2 accordingly. Such a result at least indicates that the feedback behaviors are not exclusively inherited from the daily cycle.

6.4.3 Comparison over All Workloads

After examining workloads separately, we shall investigate common feedbacks over all the workloads in Table 2.1 except LANL. Namely, we try to find whether some general

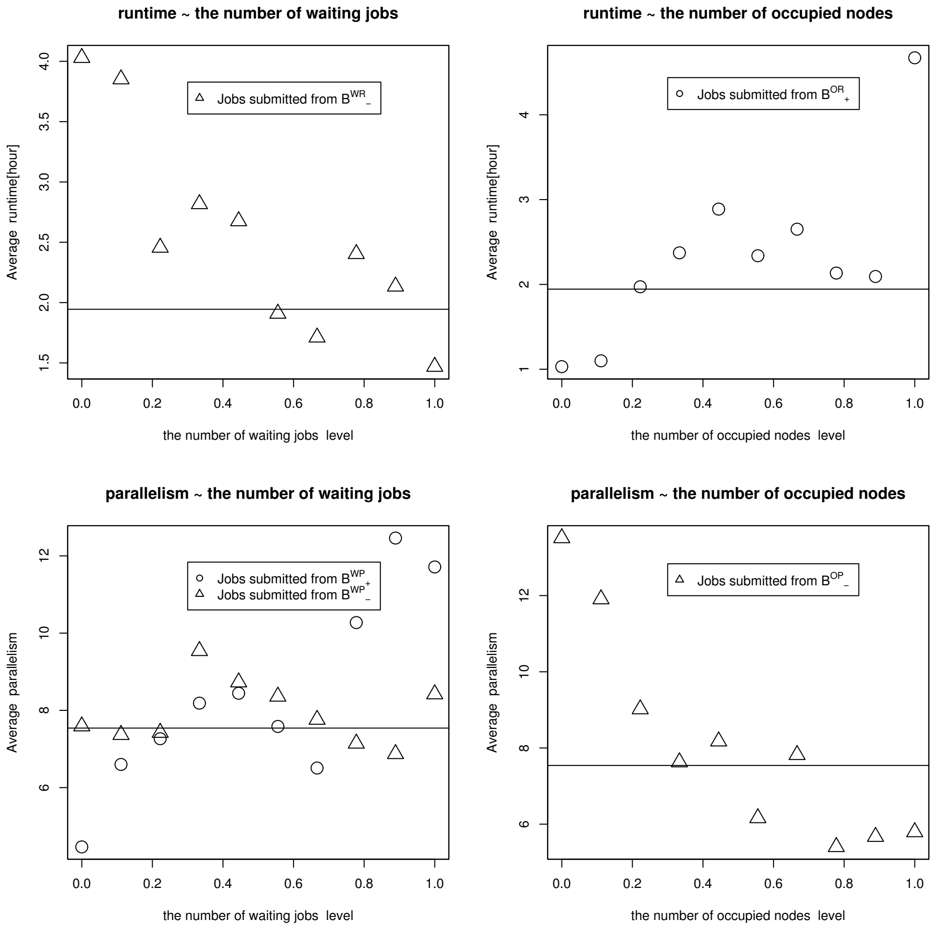

Figure 6.4: Feedback discoveries in the time frame from 1pm to 4pm in KTH

feedback phenomena exist. To this end, we summarize all the feedback results in Table 6.3. In the table, we use ”1” for the existence of a certain feedback behavior evidence and ”0” for non-existence. As we mentioned, a feedback evidence is considered in our study as the existence of the users with similar feedback behaviors and their submissions are over 20% of total submissions.

Altogether, several interesting facts can be observed from Table 6.3:

1. Certain feedbacks exist along all the workloads, including $B_-^{WR}, B_+^{OR}, B_-^{OP}$. It indicates they are rather general behaviors. That is, some users tend to submit shorter jobs when they find a longer waiting queue; some users have a tendency to submit longer jobs when many nodes are occupied; some users may deliver a job with high parallelism when they find more nodes are available.

	B_+^{WR}	B_-^{WR}	B_+^{OR}	B_-^{OR}	B_+^{WP}	B_-^{WP}	B_+^{OP}	B_-^{OP}
SDSC 96	0	1	1	1	1	1	0	1
CTC	1	1	1	0	1	0	0	1
KTH	0	1	1	1	1	0	0	1
SP2	1	1	1	1	1	0	0	1
SDSC 95	1	1	1	0	0	1	0	1

Table 6.3: Summary of discovered feedbacks in the examined workloads

2. Users with feedback B_+^{OP} are not found. It is quite reasonable that users probably would not submit the jobs with high parallelism when more nodes are busy.

3. The feedback phenomena are found in each workload, which are supported by the fact that each row have several "1"s inside. It suggests that the feedbacks are widely existing in the parallel computer environment and they should be considered for workload modeling.

6.5 Summary

In this chapter, we investigated the feedbacks of individual users. Due to lack of direct feedback factors, several implicit factors were derived from workloads. We proposed a linear model to describe how a user's jobs are affected by the feedback factors. We have found several common behaviors from the different workloads. The analysis of feedbacks can give schedulers some hints to improve the scheduling algorithms.

Until now, we have analyzed some global features in Chapter 3 and Chapter 4; we have investigated users' static and dynamic behaviors in Chapter 5 and Chapter 6. In the next chapter, we will introduce more observations about the workloads and discuss the combination of described methods to construct a comprehensive model.

Chapter 7

Discussions and Further Work

Several important aspects of workload modeling have been described in the previous chapters, including describing temporal relation, recovering missing information, clustering users and identifying feedbacks. In this chapter, we will discuss more patterns found in the workloads. Based on the analysis of different aspects of workload modeling, the combination of these aspects is discussed in order to construct a comprehensive model. The challenge for the combination is analyzed and several possible solutions are compared and discussed for our future work.

7.1 More Observations of the Workloads

Besides the discussed aspects of workload modeling, there are several interesting patterns at the global level of the workloads:

- **Daily Cycle:** The hour in a day would affect not only the number of arriving jobs but also the profile of the jobs. Figure 7.1(a) shows the daily arriving patterns of jobs. There is an obvious daily cycle: most jobs arrive during the day and only a few of them at night. Figure 7.1(b) exhibits the average runtime of jobs submitted in different hours of one day, which is completely diverse from the arriving pattern: although there are less jobs appearing at night, the runtime is longer on average. Similarly, the average parallelism is longer at night and in the morning comparing that at noon.

- **Day of Week:** We also find that less jobs are submitted during the weekend as shown in Figure 7.2(a): more jobs are submitted on weekday than on weekend. The average runtime differs significantly on weekdays in comparison to the jobs on weekends as shown in Figure 7.2(b). The runtime is longer on average during weekend than weekday.

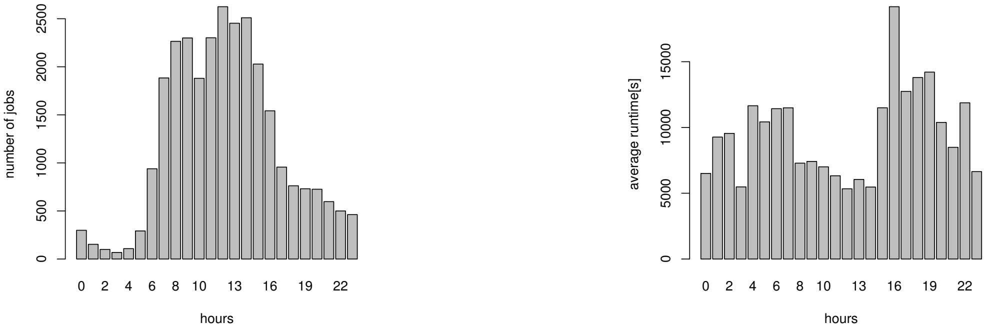

Figure 7.1: Daily cycle in KTH. Left: the number of jobs in the different hours of day. Right: the average runtime in the different hours of day

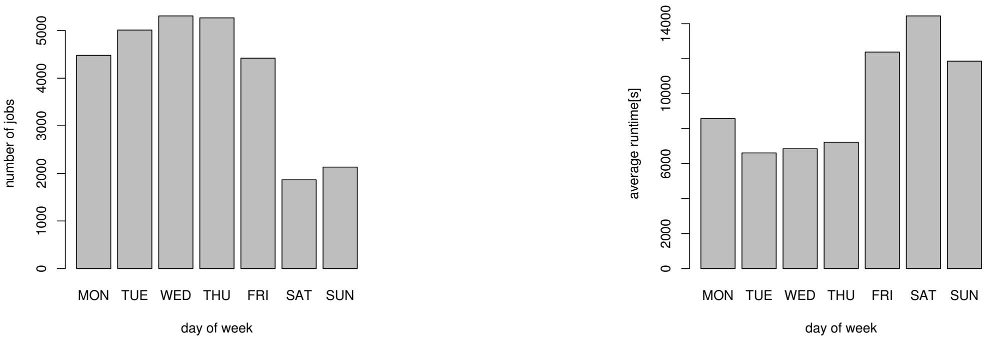

Figure 7.2: Weekday effect in KTH. Left: the number of jobs in the different days of week. Right: average runtime in the different days of week

These effects can be described by classical statistical methods. For example, Downey in [16] modeled the daily cycle using a Poisson distribution with a twist: Each day of simulated time is divided into two parts with 12 hours each. During the first part (the day) jobs arrive according to the Poisson process, but may be queued if they cannot run immediately. During the second part (the night) no more jobs arrive, and the system serves jobs that were queued during the day; Calzarossa [8] found that an eight-degree polynomial function is a suitable representation of all the analyzed arrival processes. Three representative patterns have been identified in the workloads by means of clustering applied to the coefficients of the various polynomial functions. In our work, the method based on the polynomial function is adopted due to its accuracy.

Combinations of Different Components

Another important issue in workload modeling is how these separate components can be combined to meet particular requirements posed by the specific objectives of evaluation study.

It is applicable to combine different components, when they are independent or unrelated. For instance, our correlated Markov chain in Chapter 3 and user-group model in Chapter 5 can be combined as shown in Figure 7.3: the correlated Markov chain model is used to generate synthetic jobs with temporal relations, and then the newly generated jobs are assigned to different groups, which are described by the user group model. Another example is the arriving process. It has been argued by many researchers that the profiles of jobs have no direct connection to the submission time. Therefore, the arriving process and job profiles can be considered independently: for job profiles, the correlated Markov chains can be applied, and for job arriving processes the eight-degree polynomial function can be used.

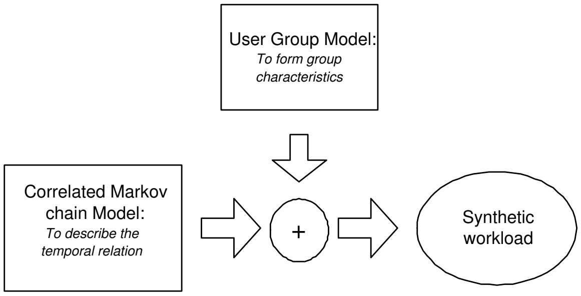
User Group Model:
To form group characteristics
Correlated Markov chain Model:
To describe the temporal relation
+
Synthetic workload

Figure 7.3: Combination of correlated Markov chain and the user group model

When the attributes of workloads are correlated, the combination task can be quite difficult. For instance, the submissions from users may not only embody the temporal correlations but also be affected by the other dynamic factors, e.g., the number of waiting jobs, as we have shown in Chapter 6. The solution can only come from the modern methodology. Here, we give several suggestions for further work.

7.2 Further Work

In our work, the combinations of different components can be regarded as constructing a complex statistical system, with multiple inputs and outputs. Consequently, we shall consider the following methods for our future research:

Artificial Neural Network

Artificial Neural network [3] is a system loosely modeled based on the human brain. It is an inherently multiprocessor-friendly architecture and without much modification. It has the ability to account for any nonlinear functional dependency. The network discovers (learns, models) the nature of the dependency without needing to be prompted. Neural network is a powerful technique to solve many real world problems. They have the ability to learn from experience in order to improve the performance and to adapt themselves to change in the environment. In addition, they are able to deal with incomplete information or noisy data. They can be very effective especially in the situations where it is not possible to define the rules or steps that lead to the solution of a problem.

In our work, the Neural network may be applied for workload modeling as shown in Figure 7.4. These potential factors e.g., the hour of day, previous runtime, are used as inputs, the job profiles like the parallelism, the runtime and the intervals are outputs of Neural network. Here, the recursive neural network is a good option to encode the feedbacks or temporal relations in workloads.

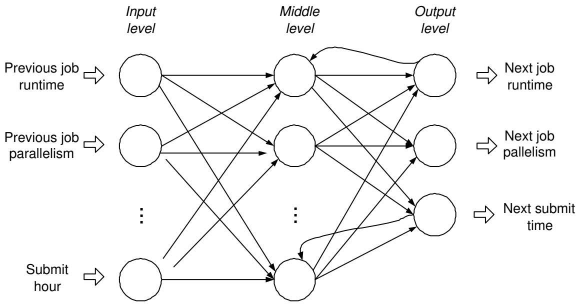

Figure 7.4: Application of Neural network for workload modeling

Support Vector Machine

A Support Vector Machine (SVM) [13] is a supervised learning technique from the field of machine learning. It is applicable to both classification and regression. Rooted in the statistical learning theory and developed by Vladimir Vapnik and co-workers at AT&T Bell Laboratories in 1995 [12], SVMs are based on the principle of structural risk minimization. An important and unique feature of this approach is that the solution is based only on those data points, which are at the margin. These points are called support vectors. The linear SVM can be extended to nonlinear one when first the problem is transformed into a feature space using a set of nonlinear basis functions. It is not necessary to implement this transformation and to determine the separating hyperplane in the possibly very-high dimensional feature space. Instead, the solution is written as a weighted sum of the values of certain kernel function.

Since the SVM can be used to simulate the complex non-linear function, it can be applied in our study as well. Furthermore, we can compare the results with the Neural network to see whether an improvement can be achieved.

Variable Length Markov chain

As we all know, the first-order Markov chain assumes first-order stationary. In other words, the model takes the consideration of one-step ahead. In many cases, the workloads can have more complicated correlations. In the future, we shall think of using more previous steps for modeling. To this end, we will choose Variable Length Markov chain (VLMC) [37] to characterize more complex cases.

VLMC is the a Markov chain with the additional structure whose memory depends on a variable number of lagged values, depending on how the actual past (the lagged values) looks like. It builds a very flexible class of tree structured models for categorical time series.

We can predict the job submissions with VLMC. Because of randomness of submission from users, the exact prediction of job submission is impracticable. Therefore, we will try to predict the range of job parameters. The prediction of job parameters is limited of help if the intervals of jobs are unknown. Therefore, the intervals will be taken into consideration in the future as well.

7.3 Summary

Workload modeling can be much more complicated than what we have analyzed until now. In this chapter, we have discussed more aspects of workload modeling, for example, daily cycle and week day effect. We also addressed the problem of the combination of different aspects into a comprehensive workload model. Finally, we provided several methods as options and gave some suggestions for our further work.

Chapter 8

Conclusion

In this work, we restricted ourselves to workload modeling for parallel computers. Previous work focused on scenarios where a large number of users contributed to the workloads. In that case, the workloads can be well described using classical statistical models, like a probabilistic distribution model. However, such a model is *not* suitable for the parallel computer environment. Due to the medium-sized user community, user-level behaviors might still be observed in the overall workloads; individual users or user groups can have important implications for scheduling system. Consequently, a new workload model is required for parallel computers. That is the focus of our study.

We started our work with examining the overall features of workloads. By analyzing the temporal relations, we found a strong sequential dependency in the job series. Therefore, we proposed a new method to correlate different Markov chains so that both the temporal dependency and the parameter correlations are considered. The results indicated that the correlated Markov chain model is capable of characterizing the temporal relation and keep the static similarity.

Another important issue is the missing information in different workloads. Because the workloads are often obtained under different resource systems, it is not uncommon that certain attributes are absent. Due to the missing attributes, the experiments or simulations based on different workload situations are not applicable. To this end, we proposed a parameterized distribution model to describe the relation between missing and existing attributes. Our results have shown that some missing information may be recovered when the relevant parameters can be trained from the complete workloads.

Based on the global-level analysis of the workloads, we investigated user-level behaviors. The detailed knowledge of individual users has crucial implications for scheduling systems. We clustered users into different groups, while each of these groups has distinct features. With these user groups, the final workloads have a direct association with individual users. The successful identification of different user groups supports our argument before: the general descriptions are not suitable for the case with the medium-size user community.

Feedback analysis is another important part in our research. Although it has been mentioned by several papers that users are adaptive to the quality of service, there are very few results about the details of feedbacks: what is the feedback? How will users react to the service of system? Our work filled this blank. Using the transformed linear model several feedback behaviors were identified, for example, some users may reduce the job runtimes when more jobs are waiting for the resources. These results indicated users have adaptive tendencies, which cannot be addressed by static models.

Our model is not a full implementation of the workload modeling tool, which requires many additional efforts. Nevertheless, the presented concept and methodology provided several features of workload modeling that are currently not available in any of initiatives. The methods in our work can be further adapted and incorporated into other tools, e.g., simulation toolkits. They may also be combined into a comprehensive modeling tool. We have given several methods as options for constructing a complete model.

All our results make it clear that: our modeling methods are capable of capturing several important aspects of parallel computer workloads. They are helpful for constructing new workload models in the future. Although the research in this work has been focused on the workload modeling for parallel computers, the concepts and methods can be applied to many applications, for instance, logistics and E-commerce.

Bibliography

[1] Kento Aida. Effect of Job Size Characteristics on Job Scheduling Performance. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, **1911**, pages 1–17. Springer Verlag, 2000. Lecture Notes in Computer Science (LNCS). 2.3

[2] Hirotugu Akaike. Information theory and an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki, editors, *Second International Symposium on Information Theory*. Akademiai Kaidó, Budapest, Hungary, 1973. 6.3.2

[3] James A. Anderson. *An Introduction to Neural Networks*. Cambridge,MA: The MIT Press, 1995. 7.2

[4] Anat Batat and Dror G. Feitelson. Gang scheduling with memory considerations. In *14th IEEE International Parallel & Distributed Processing Symposium*, pages 109–114, May 2000. 3.3.1

[5] Richard A. Becker, John M. Chambers, and Allan R. Wilks. *The New S Language*. Chapman & Hall, London, 1988. 2.4

[6] Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini, and llya Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Workshop on the Web and Databases, WebDB*, 2002. 1.1

[7] George E.P. Box and Gwilym M. Jenkins. *Time Series Analysis*. Holden Day Publishing, 1976. 3.2

[8] Maria Calzarossa and Giuseppe Serazzi. A characterization of the variation in time of workload arrival patterns. *IEEE Transactions on Computers*, **C-34**(2):156–162, Feb 1985. 7.1

[9] Walfredo Cirne and Francine Berman. A Comprehensive Model of the Supercomputer Workload. In *4th Workshop on Workload Characterization*, December 2001. 2.3, 2.3, 4.2, 4.2.1

[10] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. In *IEEE Internet Computing*, **6(1)**, pages 40–49, 2002. 1.1

[11] Jerome T. Connor, R. Douglas Martin, and L.E. Atlas. Recurrent Neural Networks and Robust Time Series Prediction. *IEEE Transactions on Neural Networks*, **5**(2):240–254, 1994. 3.2

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. **20**, pages 273–297, 1995. 7.2

[13] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, Cambridge, U.K., 2000. 7.2

[14] Morris H. DeGroot. *Probability and Statistics*. Addison-Wesley Series in Statistics. Addison-Wesley, Reading, MA, USA, 2nd edition, 1986. 3.1, 4.3.1, 4.3.2, 6.3.2

[15] Allen B. Downey. Using queue time predictions for processor allocation. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, **1291** of *Lecture Notes in Computer Science*, pages 35–57. Springer Verlag, 1997. 2.3, 2.3

[16] Allen B. Downey. A parallel workload model and its implications for processor allocation. *Cluster Computing*, **1**(1):133–145, 1998. 7.1

[17] Allen B. Downey and Dror G. Feitelson. The Elusive Goal of Workload Characterization. *ACM SIGMETRICS Performance Evaluation Review*, **26**(4):14–29, March 1999. 2.2

[18] Rakesh Dugad. A Tutorial on Hidden Markov Models and Selected Apllications in Speech Recognition. In Alex Waibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*, pages 267–296. Kaufmann, San Mateo, California, USA, 1990. 3.2

[19] Carsten Ernemann, Martin Krogmann, Joachim Lepping, and Ramin Yahyapour. Scheduling on the Top 50 Machines. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, **3277** of *Lecture Notes in Computer Science(LNCS)*, pages 17–46. Springer Verlag, October 2004. 2.1

[20] Carsten Ernemann, Baiyi Song, and Ramin Yahyapour. Scaling of Workload Traces. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, **2862** of *Lecture Notes in Computer Science (LNCS)*, pages 166–183. Springer Verlag, October 2003. 3.3.1

[21] Dror G. Feitelson. Packing Schemes for Gang Scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, **1162**, pages 89–110. Springer Verlag, 1996. Lecture Notes in Computer Science (LNCS). 2.2, 2.3, 3.1, 3.2.2

[22] Dror G. Feitelson. Workload Modeling for Performance Evaluation. In Mariacarla Calzarossa and Sara Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, **2459**, pages 114–141. Springer Verlag, 2002. Lecture Notes in Computer Science (LNCS). 2.3

[23] Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. Theory and practice in parallel job scheduling. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, **1291**, pages 1–34. Springer Verlag, April 1997. Lecture Notes in Computer Science (LNCS). 1.1, 2.2, 3.2.2

[24] Dror G. Feitelson and Ahuva M. Weil. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In *Proceedings of the 12th. International Parallel Processing Symposium*, pages 542–547, Los Alamitos, California, USA, 1998. IEEE Computer Society Press. 3.1

[25] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003. 1.1

[26] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of SuperComputer Applications*, **15**(3), 2001. 1.1

[27] Chris Fraley and Adrian E Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, **97**(458):611–631, 2002. 5.2.4

[28] Mor Harchol-Balter and Allen B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, **15**(3):253–285, 1997. 1.1

[29] Cover R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991. 2.3

[30] Joefon Jann, Pratap Pattnaik, Hubertus Franke, Fang Wang, Joseph Skovira, and Joseph Riodan. Modeling of Workload in MPPs. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, **1291** of *Lecture Notes in Computer Science (LNCS)*, pages 94–116. Springer Verlag, 1997. 1.1, 2.3, 2.3

[31] Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis.* John Wiley Sons, 1990. 5.2.2

[32] Pete Keleher, Alan L. Cox, Sandhya Dwarkadas, and Willy Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *In Proceedings of the Winter 1994 USENIX Conference*, pages 115–131, 1994. 2.4

[33] Arnaud Legrand, Loris Marchal, and Henri Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CC-Grid'03), Tokyo, Japan*, May 2003. 2.2

[34] Hubert Lilliefors. On the Kolmogorov-Smirnov Test for the Exponential Distribution with Mean Unknown. *Journal of the American Statistical Association*, **64**:387–389, 1969. 3.3.1

[35] Virginia Lo, Jens Mache, and Kurt Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, **1459** of *Lecture Notes in Computer Science (LNCS)*, pages 25–46. Springer Verlag, 1998. 1.1, 2.3, 2.3, 2.3

[36] Uri Lublin and Dror G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, **63**(20):1105–1122, 2003. 2.3, 2.3, 2.3, 2.3, 3.3.1

[37] Martin Mächler and Peter Bühlmann. Variable length markov chains. *Annals of Statistics*, **27**(2):480–513, 1998. 7.2

[38] Glenn Milligan and Martha Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, **50**(2):159–179, 1985. 5.2.2

[39] Ara V. Nefian and Monson H. Hayes. Face recognition using an embedded HMM. In *Proceedings of the IEEE Conference on Audio and Video-based Biometric Person Authentication*, pages 19–24, 1999. 3.2

[40] David M. Nichols. Implicit rating and filtering. In *Proceedings of 5th DELOS Workshop on Filtering and Collaborative Filtering*, pages 31–36. The European Research Consortium for Informatics and Mathematics, 1998. 6.1

[41] Jakob Nielsen. *Designing Web Usability: The Practice of Simplicity.* New Riders Publishing, 2000. 1.1

[42] Douglas W. Oard and Jinmook Kim. Modeling information content using observable behavior. *In Proceedings of the 64 Annual Meeting of the American Society for Information Science and Technology*, pages 38–45, 2001. 6.1

[43] Ertan Onur, Hakan Deliç, Cem Ersoy, and M. Ufuk Çaglayan. The retrial and redial phenomena in gsm networks. In *Proceedings of the IEEE Wireless Networking and Communications Conference*, pages 885–889, Chicago, Illinois, USA, September 2000. 1.1

[44] Bahl Lalit R, Frederick Jelinek, and Robert L. Mercer. A maximum likelihood approach to continuous speech recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligenc PAMI-5*, pages 179–190, 1983. Reprinted in (Waibel and Lee 1990), pp. 308–319. 3.2

[45] Rachael Rafter, Keith Bradley, and Barry Smyth. Passive profiling and collaborative recommendation. *Proceedings of the 10th Irish Conference on Artificial Intelligence and Cognitive Science, Cork, Ireland*, 1999. 6.1

[46] Emilia Rosti, Evgenia Smirni, Lawrence W. Dowdy, Giuseppe Serazzi, and Brian M. Carlson. Robust Partitioning Policies of Multiprocessor Systems. *Performance Evaluation Journal, Special Issue on Parallel Systems*, **19**(2-3):141–165, 1994. 2.2

[47] Bettina Schnor, Stefan Petri, Reinhard Oleyniczak, and Horst Langendor̈ferorfer. Scheduling of parallel applications on heterogeneous workstation clusters. In *International Conference on Parallel and Distributed Computing Systems*, Dijon, France, September 1996. 2.3

[48] Baiyi Song, Carsten Ernemann, and Ramin Yahyapour. Modeling of parameters in supercomputer workloads. In *Proceedings of the Workshop on Parallel Systems and Algorithms (PASA) in conjunction with ARCS 2004: Organic and Pervasive Computing*, **P-41** of *Lecture Notes in Informatics*, pages 400–409, March 2004. 1.1

[49] Baiyi Song, Carsten Ernemann, and Ramin Yahyapour. Parallel Computer Workload Modeling with Markov Chains. In Dror G. Feitelson, Larry

RUDOLPH, AND UWE SCHWIEGELSHOHN, editors, *Job Scheduling Strategies for Parallel Processing*, **3277** of *Lecture Notes in Computer Science (LNCS)*, pages 47–62. Springer, October 2004. 1.1

[50] BAIYI SONG, CARSTEN ERNEMANN, AND RAMIN YAHYAPOUR. User Group-based Workload Analysis and Modeling. In *Proceedings of the International Symposium on Cluster Computing and the Grid (CCGRID2005)*. IEEE Computer Society, 2005. 1.1

[51] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, September 2005. 2.4

[52] DAVID TALBY, DROR G. FEITELSON, AND ADI RAVEH. Comparing logs and models of parallel workloads using the co-plot method. In DROR G. FEITELSON AND LARRY RUDOLPH, editors, *Job Scheduling Strategies for Parallel Processing*, **1659** of *Lecture Notes in Computer Science(LNCS)*, pages 43–66. Springer Verlag, 1999. 3.3.1

[53] GAUTAMA TEMUJIN AND MARC M. VAN HULLE. Separation of Acoustic Signals Using Self-organizing Neural Networks. In *Proceedings IEEE Neural Network for Signal Processing*, pages 324–332, Wisconsin-Madison, USA, 1999. 3.2

[54] ROBERT TIBSHIRANI, GUENTHER WALTHER, AND TREVOR HASTIE. Cluster validation by prediction strength. *Journal of Computational & Graphical Statistics*, **14**(3):511–528, 2001. 5.2.2

[55] PHUOC TRAN-GIA AND MICHEL MANDJES. Modeling of customer retrial phenomenon in cellular mobile networks. *IEEE Journal on Selected Areas in Communications*, **15**(8):1406–1414, October 1997. 1.1

[56] WILLIAM N. VENABLES AND BRIAN D. RIPLEY. *Modern Applied Statistics with S. Fourth Edition*. Springer, 2002. 6.2.3

[57] JEFFREY SCOTT VITTER. An efficient algorithm for sequential random sampling. *ACM Transactions on Mathematical Software*, **13**(1):58–67, 1987. 6.3.2

[58] FANG WANG, MARIOS C. PAPAEFTHYMIOU, AND MARK S. SQUILLANTE. A gang scheduling design for multiprogrammed parallel computing environments. In DROR G. FEITELSON AND LARRY RUDOLPH, editors, *Job Scheduling Strategies for Parallel Processing*, **1162** of *Lecture Notes in Computer Science (LNCS)*, pages 111–125. Springer Verlag, 1996. 2.3

[59] YAIR WISEMAN AND DROR G. FEITELSON. Paired gang scheduling. *Transactions on Parallel and Distributed Systems*, **14**(6):581–592, Jun 2003. 3.3.1

[60] Dmitry Zotkin and Peter J. Keleher. Job-length estimation and performance in backfilling schedulers. In *IEEE International Symposium on High Performance Distributed Computing*, August. 4.1