Modeling Parallel System Workloads with Temporal Locality

Tran Ngoc Minh and Lex Wolters

Leiden Institute of Advanced Computer Science
Leiden University, 2333 CA, Leiden, The Netherlands
minhtn@liacs.nl and llexx@liacs.nl

Abstract. In parallel systems, similar jobs tend to arrive within bursty periods. This fact leads to the existence of the locality phenomenon, a persistent similarity between nearby jobs, in real parallel computer workloads. This important phenomenon deserves to be taken into account and used as a characteristic of any workload model. Regrettably, this property has received little if any attention of researchers and synthetic workloads used for performance evaluation to date often do not have locality. With respect to this research trend, Feitelson has suggested a general repetition approach to model locality in synthetic workloads [6]. Using this approach, Li et al. recently introduced a new method for modeling temporal locality in workload attributes such as run time and memory [14]. However, with the assumption that each job in the synthetic workload requires a single processor, the parallelism has not been taken into account in their study. In this paper, we propose a new model for parallel computer workloads based on their result. In our research, we firstly improve their model to control locality of a run time process better and then model the parallelism. The key idea for modeling the parallelism is to control the cross-correlation between the run time and the number of processors. Experimental results show that not only the cross-correlation is controlled well by our model, but also the marginal distribution can be fitted nicely. Furthermore, the locality feature is also obtained in our model.

1 Introduction

Parallel systems from supercomputers to clusters and grids have become more and more popular for solving many problems not only in scientific computing but also in industry. To enable effective resource allocation on these systems, numerous schedulers have been built such as Maui [5] and KOALA [15]. The quality of schedulers depends on their algorithms and has a considerable impact on the overall performance of parallel systems. Hence, the evaluation of scheduling algorithms is an essential part to build a scheduler. The accuracy of the evaluation highly relies upon workloads applied, where real workloads (called traces) are usually chosen because they reflect the system reality. However, there are several reasons showing that workload models have a number of advantages

over traces [24]. Workload modeling creates a general model which can be used to generate synthetic workloads.

Arrival time, run time and parallelism (the number of processors) are three important workload attributes necessary to be modeled to apply for studies on performance evaluation. While the arrival time attribute can be modeled individually, two remaining attributes are more difficult and require to be modeled at the same time because it is proven that there exists a cross-correlation between the run time and the parallelism [13, 24]. In [11, 23], a multifractal wavelet model is introduced to control the fractal behaviour and the temporal correlation of arrival rate processes. In [24], a combined model is suggested where the interarrival times fit a hyper-Gamma distribution and the job arrival rates match the daily cycle. Models for run time and parallelism are also proposed recently based on fitting the marginal distribution [24] or Markov chains to control the cross-correlation between these two attributes [1]. However, it can be seen that although the phenomenon of locality - a persistent similarity between nearby jobs - has been recognized to strongly exist in real parallel computer workloads [6], this important characteristic is not taken into account in the studies mentioned.

With respect to this research trend, Feitelson [6] has proposed a general repetition approach to model locality in synthetic workloads. Using this approach, Li et al. [14] recently introduced a new two-stage method for modeling the run time attribute with a temporal locality feature. The first stage consists of applying the model called *mixture of Gaussians*, whose parameters are estimated via the Model-Based Clustering (MBC) framework [3]. The second stage includes a Localized Sampling (LS) algorithm [14] for generating temporal locality in the data series. However, with the assumption that each job in the synthetic workload requires a single processor, the parallelism has not been taken into account in their model (MBC-LS). Furthermore, we also found that MBC-LS does not control the locality very well to fit the locality of real data. In this paper, we propose a new model for parallel computer workloads based on MBC-LS. In our research, we firstly improve MBC-LS to control the temporal locality feature of a run time process so that the locality of real data is matched better. Then, the parallelism is modeled with the key idea that the cross-correlation between two workload attributes -run time and parallelism- fits the real data as accurately as possible. Experimental results show that not only the cross-correlation is controlled well by our model, but also the marginal distribution can be fitted nicely. Moreover, the temporal locality feature is also obtained by improving and applying MBC-LS in our model.

The rest of this paper is organized as follows. Section 2 describes workload data used in our experiments. MBC-LS and our improvements are presented in section 3. We continue in section 4 to model the parallelism as well as to control the cross-correlation between the run time and the parallelism. Experimental results are shown in section 5. Finally, we conclude in section 6 our study and discuss some future research.

2 Workload Data

Table 1 describes details of the traces used in our study. KTH is from the IBM SP2 machine installed at the Swedish Royal Institute of Technology in Stockholm and is scheduled using the EASY backfilling scheduler [4]. LANL is from the Connection Machine CM-5 installed at Los Alamos National Lab and is scheduled using DJM [8]. SDSC is collected from the San Diego Supercomputer Center Intel Paragon machine whose scheduling is based on NQS [16]. This trace is available under two separate one-year traces, namely SDSC95 and SDSC96. Though the first four traces in Table 1 are rather old, we select them in our experiments for comparison with recent studies [1, 24].

Table 1. Traces used in the experiments.

Trace	Period	Number of processors	Number of jobs
KTH	09/1996-08/1997	100	28480
LANL	01/1996-09/1996	1024	37517
SDSC95	12/1994-12/1995	416	53885
SDSC96	12/1995-12/1996	416	32032
LLNLATLAS	02/2007-05/2007	9216	23911
GRID5000	07/2006-08/2006	3216	42171

Newly collected traces are also considered in our study, including LLNLATLAS and GRID5000. At the cluster level, LLNLATLAS, a large Linux cluster called Atlas installed at the Lawrence Livermore National Lab and scheduled by MOAB [18], is selected. At the grid level, GRID5000 [10], consisting of 9 sites with a total of 15 clusters geographically distributed in France, is chosen. GRID5000 uses OARGrid as grid resource broker and OAR as batch scheduler for its local clusters [19]. Note that this grid trace includes both jobs submitted via the grid resource broker and jobs directly submitted to the clusters. All traces and detailed information are available on [21], except for GRID5000 which are collected from [9].

3 Modeling Job Run Time with Locality

We first show in this section the reason why we need to take into account temporal locality when modeling the run time attribute. Then, we present briefly a two-stage approach [14] to model job run time with the locality feature. The first stage consists of the *mixture of Gaussians* model, whose parameters are estimated via Model-Based Clustering (MBC) framework. The second stage includes a Localized Sampling (LS) algorithm for generating temporal locality in the data series. In addition, we also describe our improvement on this approach to control locality better.

3.1 Why Locality?

In the effort of improving the performance of parallel systems, several researches on predicting job run time using historical data to provide schedulers with better information have been done [12, 22, 26]. These studies are based on the belief that the recent past is indicative of the near future. It is a generalization of the idea of locality. Furthermore, we observe that the real workload data is far from independently and identically distributed, instead, similar jobs tends to arrive within bursty periods. Therefore, locality should be taken into account when modeling parallel system workloads to help studies on predicting more accurately.

3.2 Model-Based Clustering

Model-Based Clustering (MBC) [3] is a methodological framework that underlies a powerful approach not just to data clustering but also to discriminant analysis and multivariate density estimation. Instead of looking for a single probability density function for the distribution of the data, the main idea of MBC is that it considers the data as generated by a mixture of normal (Gaussian) probability density functions, where each function represents a different cluster[1]. The selection of the number of clusters is based on the Bayesian information criterion. Gaussian parameters for these clusters are calculated by combining agglomerative hierarchical clustering and the expectation-maximization algorithm for maximum likelihood. MBC is implemented in the MCLUST software and available on [17].

3.3 Localized Sampling

The localized sampling algorithm presented in this section is used to model the locality feature of a job run time process. The concept and phenomenon of temporal locality [20] has been recognized and recently studied to model parallel workloads. To this end, a locality of sampling algorithm has been introduced by Feitelson [6] based on the observation that the lengths of repetitions of equivalent jobs empirically follow a Zipf-like (power law) distribution [7]. This algorithm can be summarized by the following steps:

1. Sample a variate X from the probability distribution of the data.
2. Sample a variate R from the fitted Zipf distribution of repetitions in the data.
3. Repeat the X variate R times to distort the distribution locally.
4. Return step 1 until the desired number of samples has been generated.

[1] The term "cluster" stems from the concept of "data clustering". Data clustering is the classification of similar objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait. (definition from Wikipedia)

Though the locality can be obtained well, this algorithm still has a number of limitations. Firstly, it is not easy to know exactly the probability density function of the real data. Hence, sampling for the variate X in step 1 is very difficult and almost infeasible. Secondly, repetition of a single value in step 3 is a simple treatment and more sophisticated techniques are needed for better stochastic approximation. To this end, a localized sampling algorithm has been proposed by Li et al. in [14] to overcome these limitations. They found that not only the repetitions of real data but also the repetitions of cluster labels empirically follow a power law. The classification/cluster labels can be obtained via MBC presented in section 3.2. The idea of their algorithm is that the cluster label series instead of the real data is taken into account to serve as the input for the 4-step procedure above. In this way, the first limitation is solved because the probability distribution of cluster labels is known in advance via MBC. Futhermore, the second limitation is also eliminated because each cluster label represents a cluster of values instead of a single value. Each cluster label in the generated series is converted into a specific value in the final synthetic run time process by sampling the distribution of the corresponding cluster, which is also known in advance via MBC.

Although using MBC to classify data series and applying these classifications to the 4-step procedure above is a good idea, we found that a new limitation occurs with this approach. That is only repetitions of cluster labels follow the power law, but repetitions of the final synthetic run time process do not fit the Zipf-like distribution as the real data any more. It is because the authors use a cluster of values instead of a single value to overcome the second limitation in Feitelson's algorithm. Therefore, we propose a solution to solve this problem. When we repeat a cluster label R times, we also equivalently sample the distribution of that cluster R times to produce R specific values in the final synthetic run time process. Our idea is that instead of sampling the distribution of a cluster R times, we will produce a single value r times with $r < R$ and then sample the distribution of the cluster $R - r$ times. So how to obtain the value r? Observing all the times where a cluster label appears repeatedly in the cluster label series, we recognize that there are periods where no single value is produced repeatedly. Based on this observation, we calculate a probability p to indicate the ability that there is a single value produced repeatedly at a repetition time of a cluster label. As such, r can be calculated by sampling from the fitted Zipf distribution of repetitions in the real data with a probability p. Otherwise, with a probability $1 - p$, r is assigned to be equal to 0.

Combining the idea of Feitelson, the idea of Li et al. and our idea, we summarize the following steps to model the locality feature for a run time process:

1. Run the model-based clustering procedure in section 3.2, where the real run time process $Data_i, i = 1 \rightarrow n$ serves as an input, to obtain *mixture of Gaussians* parameters $(\mu_k, \sum_k; p_k), k = 1 \rightarrow G$ and classifications $L_i \in \{1, \cdots, G\}, i = 1 \rightarrow n$; where G is the number of clusters, μ_k, $\sum_k$ and p_k are mean, variance and probability of cluster k, respectively.

2. Count the lengths of repetitions in L_i and fit them to a Zipf distribution Z_L. For example, if $L_i = \{2, 2, 2, 3, 1, 1, 4, 5, 5, 5, 5\}$, we have a series of lengths of repetitions as $\{3, 1, 2, 1, 4\}$ and fit this series to a Zipf distribution.
3. Count the lengths of repetitions in $Data_i$ and fit them to a Zipf distribution Z_D.
4. Calculate the probability p for the occurence of the repetition of a single value within each repetition in classifications L_i.
5. Generate a series of cluster labels C according to the cluster probability p_k.
6. Set the window size W. Form a series C_σ by applying the cluster permutation procedure[2] . This step is used to control the autocorrelation in the synthetic data and completely independent on the locality. The autocorrelation increases when W is large and in the simplest case, $C_\sigma = C$ when $W = 1$. This step can be bypassed without any impact on the locality by simply setting $W = 1$ if users do not want to control the autocorrelation.
7. Select a cluster label c from C_σ sequentially.
8. Sample a variate R from the fitted Zipf distribution Z_L.
9. Sample a variate $Prob$ from the uniform distribution over the range $[0, 1]$.
10. If $Prob \leq p$, sample a variate r from the fitted Zipf distribution Z_D, else assign $r = 0$. Note that the sampling work is done by a loop until we obtain $r < R$.
11. Sample the Gaussian distribution $f_c(\mu_c, \sum_c)$ to obtain a single value and repeat this value r times.
12. Sample the Gaussian distribution $f_c(\mu_c, \sum_c)$ $R - r$ times.
13. Return step 7 until the desired number of samples has been generated.

4 Modeling Parallelism and Control the Cross-Correlation

For most parallel systems, parallelism is another vital workload attribute beside run time. Furthermore, the cross-correlation between it and the run time is also very important. In [25], Lo et al. demonstrated how different degrees of this cross-correlation might lead to discrepant conclusions about the evaluation of scheduling performance. Therefore, we should take into account this cross-correlation when modeling parallel system workloads.

As indicated in [14], despite the fact that the *mixture of Gaussians* model is a good choice for fitting the marginal distribution, it is not suitable for some attributes with discrete values such as parallelism. Hence, we propose in this section a new three-stage approach to model the parallelism as well as control the cross-correlation between it and the run time. Firstly, the parallelism process is classified into a number of classes. However, different from the run time process with continuous values, the parallelism process with discrete values can not be classified via MBC in section 3.2. Rather than, we create a new method for the classification of the parallelism process. Secondly, we control the cross-correlation

[2] Hereby we give an example of the cluster permutation procedure, for details see [14]. If we have a series of cluster labels generated in step 5 $C = \{1, 2, 1, 3, 2, 2, 3, 2, 4, 1, 4\}$ and $W = 4$, we deduce $C_\sigma = \{1, 1, 2, 3, 2, 2, 2, 3, 4, 4, 1\}$.

between the run time and the parallelism by creating and using a transition table. Thirdly, we convert class labels into specific values based on the sample probability.

4.1 Classify the Parallelism

Algorithm 1 Classify the parallelism process. The operator $length(\cdot)$ indicates the length of a series and the operator $round(X)$ rounds X to the nearest integer.

Input: A parallelism process $P_i, i = 1 \rightarrow n$.
Output: A classification process $C_i, i = 1 \rightarrow n$ where C_i indicates the class to which P_i belongs.

Assign $maxcpus = max(\{P_i, i = 1 \rightarrow n\})$;
for $j = 1$ to $maxcpus$ **do**
  Calculate the number of occurences of j in $\{P_i\}$: $count_j = length(\{x = j, x \in \{P_i\}\})$;
  if $count_j \neq 0$ **then**
    $count_j = round(log_2(count_j)) + 1$;
  end if
end for
for $i = 1$ to n **do**
  $C_i = count_{P_i}$;
end for

Our approach to classify the parallelism is presented in detail in Algorithm 1. We start by grouping jobs that require the same number of processors and count the number of jobs in each group. Then, each group is assigned a label which is an integer calculated by rounding the logarithm of the number of jobs in that group to the base 2 and adding 1. Jobs belong to a group are also classified with its label. As such, groups that have approximately equal quantities of jobs will be assigned the same label. For example, if there are 250 jobs requesting 4 cpus and 300 jobs requesting 10 cpus, all of them will be classified as 9. Note that this classification approach can only be applied on a series with discrete values such as parallelism.

4.2 Control the Cross-Correlation

We use Algorithm 2 to control the cross-correlation between the run time and the parallelism. Firstly, we calculate the transition conditional probability table $Pr(c, l)$, where c and l are labels of the parallelism and the run time, respectively. $Pr(c, l)$ indicates the probability for a job to have the parallelism label c with the condition that the label for its run time is known in advance as l. $Pr(c, l)$ of a job is calculated by the ratio between the probability $P(c, l)$ for that job to have the parallelism label c and the run time label l at the same

Algorithm 2 Create a series of parallelism labels.

Input: Classifications of the run time process $L_i, i = 1 \rightarrow n$ obtained via MBC in section 3.2, classifications of the parallelism process $C_i, i = 1 \rightarrow n$ obtained via Algorithm 1 and the series of cluster labels CW.
Output: A series of parallelism labels CL.

Assign $maxruntimelabel = max(\{L_i, i = 1 \rightarrow n\})$;
Assign $maxcpulabel = max(\{C_i, i = 1 \rightarrow n\})$;
for $l = 1$ to $maxruntimelabel$ **do**
  $P(l) = \frac{length(\{x=l, x \in L_i\})}{n}$;
end for
for $c = 1$ to $maxcpulabel$ **do**
  for $l = 1$ to $maxruntimelabel$ **do**
    $P(c, l) = \frac{length(\{i \in [1,n]: C_i = c, L_i = l\})}{n}$, where i represents a job;
    $Pr(c, l) = \frac{P(c,l)}{P(l)}$;
  end for
end for
for each cw_j in CW **do**
  Select an integer $x \in [1, maxcpulabel]$ according to the transition conditional probability table $Pr(c, l)$ with $l = cw_j$ and assign $cl_j = x$ to form a series of parallelism labels CL;
end for

time and the probability $P(l)$ for that job to have the run time label l. Secondly, we form a series of parallelism labels based on the transition probability table $Pr(c, l)$. Each parallelism label corresponds to a cluster label in the series CW. We obtain CW by using C_σ and repeating each cluster label in C_σ R times. Reminding that in the algorithm presented in section 3.3, C_σ is formed in step 6 by applying the cluster permutation procedure. Each cluster label from C_σ is selected and repeated R times in step 7 and step 8, where R is sampled from the fitted Zipf distribution. For example, if we have $C_\sigma = \{1, 3, 2\}$ and the values of R for these labels are $2, 1, 4$ respectively, we obtain $CW = \{1, 1, 3, 2, 2, 2, 2\}$.

4.3 Generate Specific Values

This stage receives a series of run time labels CW and a series of parallelism labels CL as its inputs. The way to obtain CW is presented in section 4.2. CL can be achieved via Algorithm 2. Combining CW and CL, we have a series of labels for parallel jobs (cw_i, cl_i), where $cw_i \in CW$ and $cl_i \in CL$. The specific value for the run time of job i is generated by sampling the distribution of the cluster with label cw_i. The specific value for the number of processors of job i with label cl_i is generated by the following steps:

1. Determine all jobs in the real data that have labels of (cw_i, cl_i) based on classifications of the run time process $L_i, i = 1 \rightarrow n$ obtained via MBC in section 3.2 and classifications of the parallelism process $C_i, i = 1 \rightarrow n$

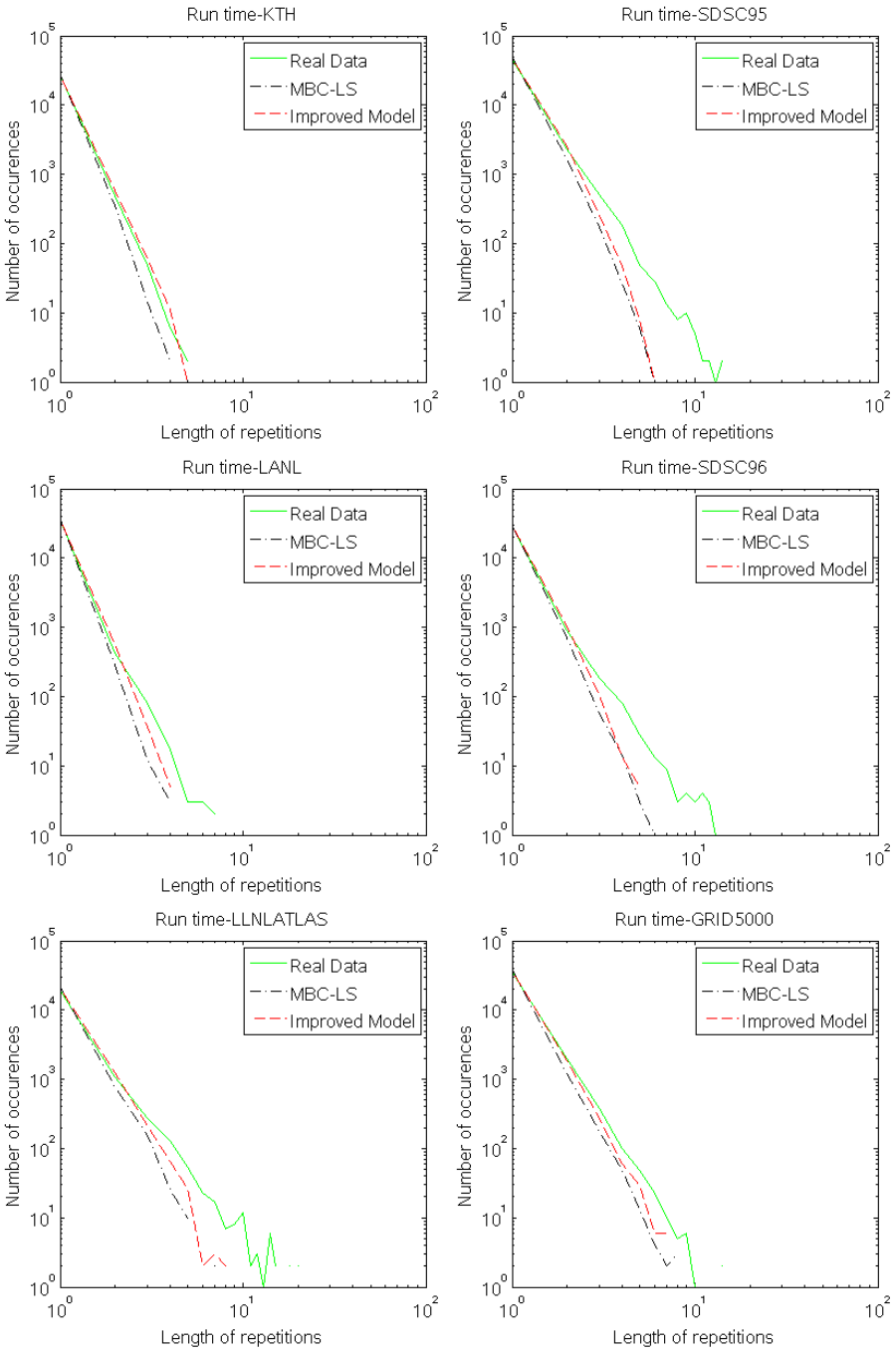

Fig. 1. Log-log histograms for the lengths of repetitions in the traces and synthetic run time processes.

obtained via Algorithm 1. We can know exactly and call the number of processors of these jobs $\{processors\}$.
2. Consider $\{processors\}$ as a sample space, the specific value we seek is selected in $\{processors\}$ according to the uniform probability of this space.

5 Experimental Results

Details of the traces used in our experiments are described in section 2. All these traces are applied on our model to generate synthetic workloads. The quality of these synthetic workloads is evaluated by comparing with the real data. Furthermore, we compare our model with the model of Song et al. [1] and the model of Lublin/Feitelson [24]. They are recent models for parallel system workloads.

Evaluation metrics used in our experiments include the marginal distribution, the cross-correlation between the run time and the parallelism, and the *squashed area* discussed in section 5.2. Note that we do not evaluate the locality feature of the run time process since it is assured by the Zipf distribution of repetitions. Instead, we only evaluate our improvement by comparing with MBC-LS model [14].

5.1 Locality of the Run Time Process

Figure 1 shows experimental results in evaluating our improvement. Reminding that our purpose is to fit the locality of the real data better. In order to compare our model with MBC-LS, we count the lengths of repetitions in both synthetic and real run time processes and draw the log-log histograms of these lengths. It can be seen that in all cases, our improved model fits the real data better than MBC-LS. Nevertheless, the model does not match the real data very well. It is because the probability p we calculate in step 4 of the algorithm presented in section 3.3 is not a perfect value. Another reason is that we only allow one sequence of r repeated values each time R values are generated. Of course, we can improve this matching by increasing p and allow more than one sequence but it depends on the traces. In our experiment, we found that this method indeed helps to match some traces better but also causes the situations of overfitting for other traces. Therefore, we decide to select the current method to avoid overfitting. However, more research is still left to improve the locality matching of the model compared with the real data.

5.2 The Metric Squashed Area

The *squashed area* (SA) metric is proposed by Song et al. [1]. It is the total resource consumptions of all jobs

$$SA = \sum_{j \in Jobs} req\_processor_j \times run\_time_j. \quad (1)$$

Furthermore, the difference of *squashed area* is calculated by

$$d\_SA = \frac{synthetic\_SA - original\_SA}{original\_SA}. \quad (2)$$

Table 2. The difference of *squashed area*. Results for Song et al.'s model are collected from [1].

Trace	Our model	Song et al.
KTH	0.39%	15%
LANL	2.21%	-1%
SDSC95	-0.04%	-3%
SDSC96	-3.12%	8%

In [2], Ernemann et al. demonstrated that the *squash area* has significant impacts on scheduling performance. It can be concluded from Equation (2) that if d_SA is closer to 0, the result is better (i.e. the model matches well the real traces). The results in Table 2 show that our model is better than the model of Song et al. since in most cases our differences of *squashed area* are smaller.

5.3 The Metric Cross-Correlation

One of the most difficult problems in modeling parallel workloads is how to control the cross-correlation between the run time and the parallelism as accurately as in the real data. The cross-correlation is measured by calculating the correlation coefficient between the run time and the parallelism. It can be seen from Table 3 that our model controls the cross-correlation well since our results are closer to the real data than those of the other models. As understanding from our experiments, the cross-correlation is controlled well thanks to the combination of Algorithm 2 and the way we generate specific values for parallelism labels as described in section 4.3.

Table 3. The cross-correlation between the run time and the parallelism. Results for the models of Song et al. and Lublin/Feitelson are collected from [1].

Trace	Real data	Our model	Song et al.	Lublin/Feitelson
KTH	0.011	0.015	0.005	0.005
LANL	0.172	0.192	0.226	0.29
SDSC95	0.277	0.233	0.140	0.105
SDSC96	0.371	0.332	0.155	0.116
LLNLATLAS	0.034	0.033	-	-
GRID5000	0.006	0.009	-	-

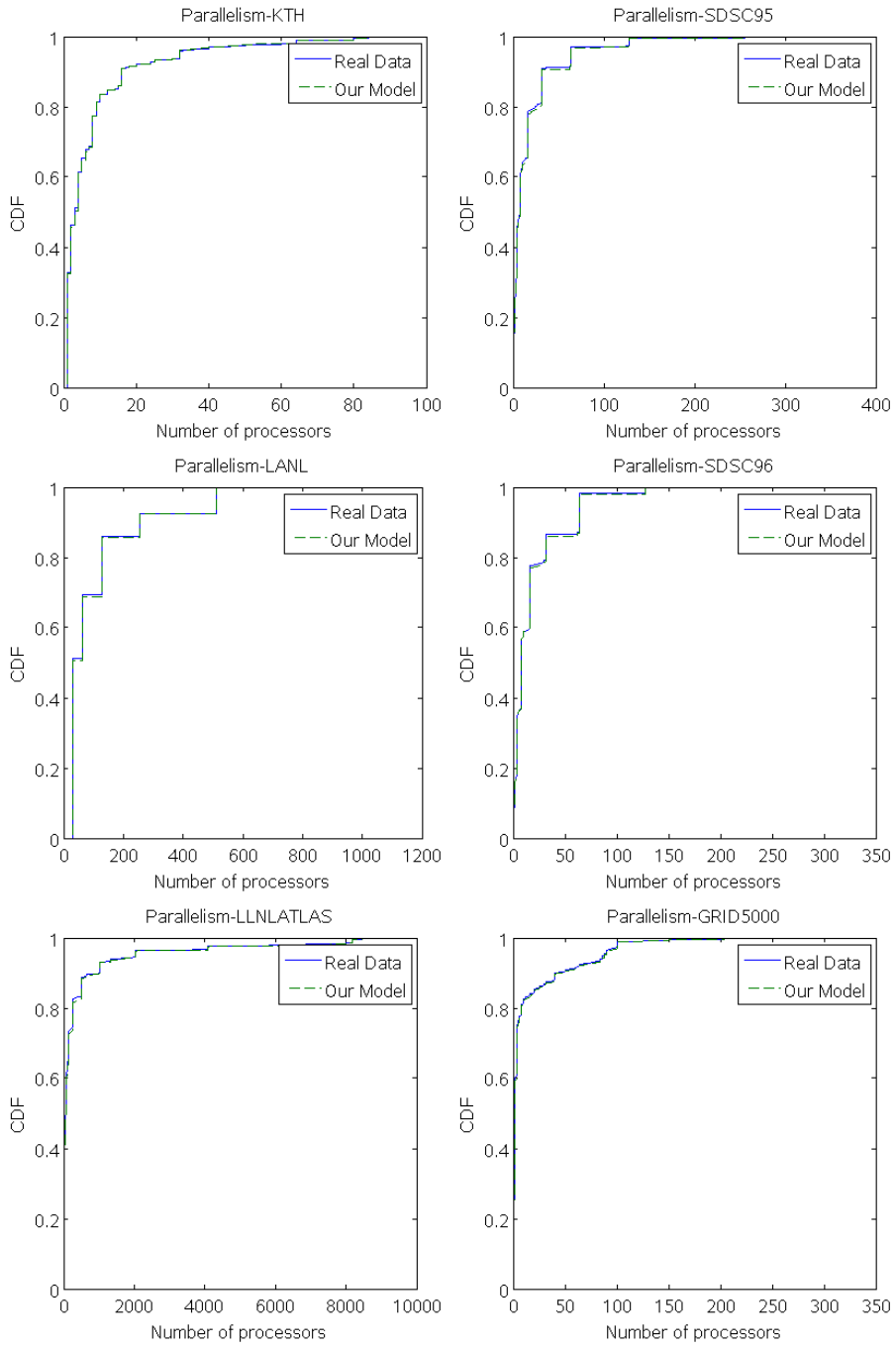

Fig. 2. Fitted marginal distribution of the parallelism.

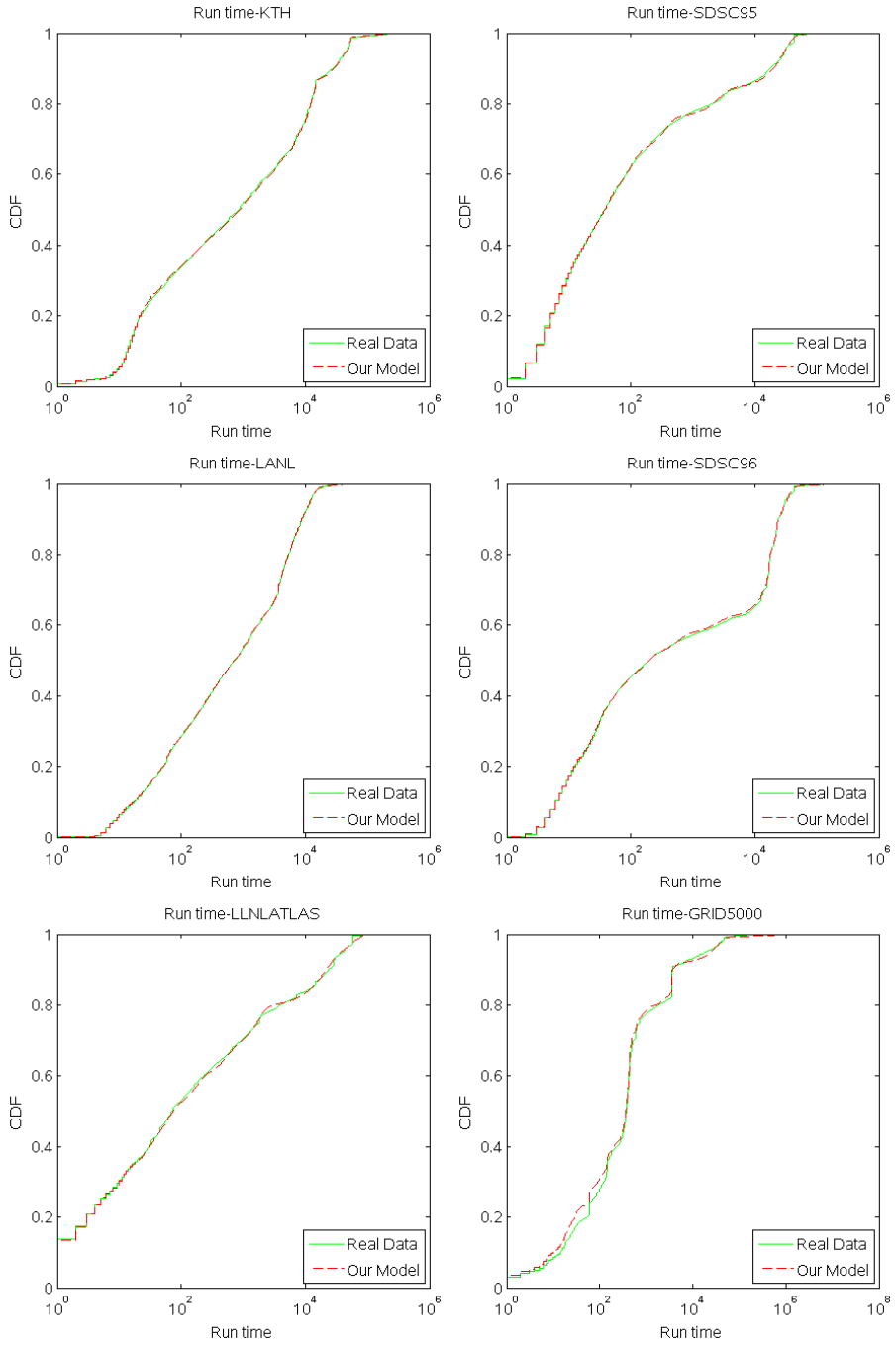

Fig. 3. Fitted marginal distribution of the run time.

5.4 Marginal Distribution

Another important result from our model is that the marginal distribution is fitted very well. Figure 2 and Figure 3 show how well the cumulative density function (CDF) of the run time and the parallelism is fitted in our model. For the run time with continuous values, the marginal distribution is determined by the *mixture of Gaussians* model (see section 3.2). For the parallelism with discrete values, our experiment proves that the marginal distribution is fitted well by Algorithm 1 in section 4.1.

6 Conclusions and Future Work

When modeling parallel system workloads, researchers should take care of the locality and the cross-correlation between the parallelism and the run time. The locality feature is necessary for studies on predicting job run time, based on the belief that the recent past is indicative of the near future. The cross-correlation was demonstrated in [25] to have significant impacts on the evaluation of scheduling performance.

With respect to the locality, Li et al. [14] and Feitelson [6] recently introduced approaches to produce locality in the synthetic run time process. We also discussed some limitations of their methods and suggested a solution to overcome these limitations. Our solution indeed fits locality of the real data better than Li et al. 's model (see Figure 1) but not very well. The reason was already discussed in section 5.1 and more effort to improve this result is left for future.

For the cross-correlation, experimental results (see Table 2 and Table 3) showed that our model can control the cross-correlation between the run time and the parallelism more accurately, compared with recent models for parallel system workloads [1, 24].

In addition, another important result from our model is that the marginal distributions of the synthetic run time and the synthetic parallelism fit the real data very well (see Figure 2 and Figure 3).

From our results, we believe that modeling parallel system workloads based on classifying data is a good approach. In future work, we continue to use this idea to model other workload attributes such as user estimated run time and requested memory in order to form a full synthetic workload with adequate necessary attributes including real run time, user estimated run time, requested memory and parallelism.

References

1. B. Song, C. Ernemann, R. Yahyapour, "Parallel Computer Workload Modeling with Markov Chains", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 3277, Pages 47-62, 2004.
2. C. Ernemann, B. Song, R. Yahyapour, "Scaling of Workload Traces", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 2862, Pages 166-182, 2003.

3. C. Fraley, A. E. Raftery, "Model-Based Clustering, Discriminant Analysis, and Density Estimation", Journal of the American Statistical Association, Volume 97, Pages 611-631, 2002.
4. D. A. Lifka, "The ANL/IBM SP Scheduling System", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 949, Pages 295-303, 1995.
5. D. B. Jackson, Q. Snell, M. J. Clement, "Core Algorithms of the Maui Scheduler", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 2221, Pages 87-102, 2001.
6. D. G. Feitelson, "Locality of Sampling and Diversity in Parallel System Workloads", in Proceedings of 21st ACM International Conference on Supercomputing, ACM Press, USA, 2007.
7. D. G. Feitelson, "Workload Modeling for Computer Systems Performance Evaluation". Book Draft, Version 0.18, 2008.
8. Distributed Job Manager, http://bradley.c-sail.mit.edu/cm5docs/manuals/cm5/doc/djm/.
9. Grid Workloads Archive, http://gwa.ewi.t-udelft.nl/.
10. Grid5000, http://www.grid5000.org/.
11. H. Li, "Long Range Dependent Job Arrival Process and Its Implications in Grid Environments", in Proceedings of MetroGrid Workshop, 1st International Conference on Networks for Grid Applications, ACM Press, France, 2007.
12. H. Li, D. Groep, L. Wolters, " An Evaluation of Learning and Heuristic Techniques for Application Run Time Predictions", in Proceedings of 11th Annual Conference of the Advance School for Computing and Imaging (ASCI), Netherlands, 2005.
13. H. Li, D. Groep, L. Wolters, "Workload Characteristics of a Multi-Cluster Supercomputer", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 3277, Pages 176-193, 2005.
14. H. Li, M. Muskulus, L. Wolters, "Modeling Correlated Workloads by Combining Model Based Clustering and a Localized Sampling Algorithm", in Proceedings of 21st ACM International Conference on Supercomputing, ACM Press, USA, 2007.
15. H. Mohamed, D. Epema, "The Design and Implementation of the KOALA Co-Allocating Grid Scheduler", European Grid Conference, Lecture Notes in Computer Science, Volume 3470, Pages 640-650, 2005.
16. M. Wan, R. Moore, G. Kremenek, K. Steube, "A Batch Scheduler for the Intel Paragon with a Non-Contiguous Node Allocation Algorithm", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 1162, Pages 48-64, 1996.
17. MCLUST, http://www.stat.washington.edu/mclust/.
18. Moab Workload Manager, http://www.clusterresources.com/pages/products/moab-cluster-suite/workloadmanager.php.
19. OAR, http://oar.imag.fr/.
20. P. J. Denning, "The Locality Principle", Communications of ACM, Volume 48, Pages 19-24, 2005.
21. Parallel Workloads Archive, http://www.cs.h-uji.ac.il/labs/parallel/workload/.
22. R. Gibbons, "A Historical Application Profiler for Use by Parallel Schedulers", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 1291, Pages 58-77, 1997.
23. R. H. Riedi, M. S. Crouse, V. J. Ribeiro, R. G. Baraniuk, "A Multifractal Wavelet Model with Application to Network Traffic", Journal of IEEE Transactions on Information Theory, Volume 45, Issue 4, Pages 992-1018, 1999.

24. U. Lublin, D. G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs", Journal of Parallel and Distributed Computing, Volume 63, 2003.
25. V. Lo, J. Mache, K. Windisch, "A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 1459, Pages 25-46, 1998.
26. W. Smith, I. Foster, V. Taylor, "Predicting Application Run Times Using Historical Information", Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Volume 1459, Pages 122-142, 1998.