The handle http://hdl.handle.net/1887/18620 holds various files of this Leiden University dissertation.

**Author:** Tran, Minh Ngoc  
**Title:** Workload modeling and performance evaluation in parallel systems  
**Issue Date:** 2012-03-22
Chapter 7

Performance Impact of Runtime and Parallelism on Scheduling

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

7.1 Control Correlation

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

![Diagram](image)

**Figure 7.1:** Illustration of controlling correlation.

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

\(^2\)In Figure 7.1, each element in one process will associate with one element of the other process, showing by either a continuous or discrete two-direction arrow. We use the words “partner” and “partnership” to indicate the associated element and the association between two elements, respectively.
in Figure 7.1 exhibit a negative correlation, we have \( d_j = -0.6 \) because 60% elements are not permuted. The outputs of the procedure consist of \( m \) new parallelism processes \( \{ P^j_i \}, j = 1, \ldots, m \). Each \( \{ P^j_i \} \) associates with \( \{ R_i \} \) to yield a correlation that corresponds to \( d_j \).

**Algorithm 3** Procedure *Control_Correlation*.

**Input:** a runtime process \( \{ R_i \} \), a parallelism process \( \{ P_i \} \) and a list of \( m \) parameters \( d_j, j = 1, \ldots, m \).

**Output:** \( m \) new parallelism processes \( \{ P^j_i \}, j = 1, \ldots, m \).

// Maximize the correlation
\[
\{ R'_i \} = \text{sort}(\{ R_i \}', \text{ascend}');
\{ P_{p_i} \} = \text{sort}(\{ P_i \}', \text{ascend}');
\{ P_{n_i} \} = \text{sort}(\{ P_i \}', \text{descend}');
\]

// To assure that locality is not affected
Rearrange \( \{ R'_i \} \) to \( \{ R_i \} \) while fixing its partnerships with \( \{ P_{p_i} \} \) and \( \{ P_{n_i} \} \);

// Generate \( m \) parallelism processes and tune correlation
for \( j = 1 \) to \( m \) do
    if \( d_j \geq 0 \) then
        \( \{ P^j_i \} = \text{PERMUTATION}(\{ P_{p_i} \}, 1-d_j) \);
    else
        \( \{ P^j_i \} = \text{PERMUTATION}(\{ P_{n_i} \}, 1+d_j) \);
    end if
end for

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

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

\(^3\)In theory when a correlation reaches the maximal threshold, the correlation coefficient reaches \( \pm 1 \). The Spearman’s rank correlation coefficient can only obtain \( \pm 1 \) in case there are no tied ranks, i.e., elements of a stochastic process must have distinguish values so that they do not have the same ranks [60, 75]. However, tied ranks exist in real runtime and parallelism processes due to repeated data values. Therefore, the maximal correlation that can be achieved in practice is often only approximately equal to \( +/−1 \), which is refered as a maximal positive/negative correlation, respectively.
use Spearman’s approach which is based on data ranks instead of the data itself to calculate a correlation coefficient. Hence to maximize the correlation coefficient, we sort \( \{R_i\} \) in an ascending order to form \( \{R'_i\} \) and sort \( \{P_i\} \) in ascending and descending orders to form \( \{Pp_i\} \) and \( \{Pn_i\} \), respectively. Consequently, we achieve a maximal positive correlation between \( \{R'_i\} \) and \( \{Pp_i\} \), and a maximal negative correlation between \( \{R'_i\} \) and \( \{Pn_i\} \). As the second step, we rearrange \( \{R'_i\} \) to \( \{R_i\} \) since our purpose is to keep the input runtime process unchanged to ensure the procedure will not affect locality. In this rearrangement, we keep the partnerships between elements of \( \{R'_i\} \) and elements of \( \{Pp_i\} \) and \( \{Pn_i\} \) unchanged, i.e., whenever we swap 2 elements of \( \{R'_i\} \), we also swap their partners in \( \{Pp_i\} \) and \( \{Pn_i\} \). After the rearrangement, \( \{R'_i\} \) turns back to \( \{R_i\} \) which now has maximal positive and negative correlations with \( \{Pp_i\} \) and \( \{Pn_i\} \), respectively.

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

![Figure 7.2: The relationship between correlation and \( d_j \), shown for the trace HPC.](image)

The last problem we address in this section is the meaning of the list of parameters \( \{d_j\} \). As explained above, \(|d_j|\) indicates which percentage of elements in a stochastic process is kept unchanged by the PERMUTATION function, and the sign of \( d_j \) indicates a positive/negative correlation. However, we would like to know whether \( d_j \) has any relation with the degree of correlation. To check this, we do an experiment by applying the procedure \textit{Control Correlation} on the runtime and the parallelism pro-
cesses of the trace HPC with the list of parameters \( \{d_j\} = \{-1, -0.9, -0.8, \ldots, +0.8, +0.9, +1\} \). Interestingly, the result in Figure 7.2, which draws the Spearman’s correlation as a function of \( d_j \), shows that there exists a linear relationship between \( d_j \) and the degree of correlation. When fitting the line in Figure 7.2, we find that its slope is approximately equal to the maximal correlation that HPC can obtain via the first step of the procedure. Control Correlation: \( | \pm 0.95 | \). In other words, the line is of the form \( \text{Correlation} = |\text{MaxCor}| \times d_j \), where \( \text{MaxCor} \) is the maximal correlation obtained in the first step of the procedure. As such, when the possible maximal correlation of a trace reaches \( \pm 1 \), we can consider \( d_j \) as the degree of correlation. Mathematically, this consequence can be proven as follows. Given two \( N \)-element stochastic processes \( X \) and \( Y \) which exhibit a maximal correlation \( \text{MaxCor} \). With a parameter \( d_j \in [-1, +1] \), we select elements randomly to divide \( X, Y \) correspondingly into \([1 - |d_j|] \times N\)-element processes \( X_1, Y_1 \) and \([|d_j|] \times N\)-element processes \( X_2, Y_2 \). In statistics, \( X_1, X_2 \) are considered two samples of the population \( X \), and \( Y_1, Y_2 \) are two samples of the population \( Y \). As such, we have \(^4\)

\[
\begin{align*}
\bullet & \quad \mu_{X_1} \simeq \mu_{X_2} \simeq \mu_X \quad \text{and} \quad \mu_{Y_1} \simeq \mu_{Y_2} \simeq \mu_Y , \\
\bullet & \quad \sigma_{X_1} \simeq \sigma_{X_2} \simeq \sigma_X \quad \text{and} \quad \sigma_{Y_1} \simeq \sigma_{Y_2} \simeq \sigma_Y , \\
\bullet & \quad \rho_{X_1,Y_1} \simeq \rho_{X_2,Y_2} \simeq \rho_{X,Y} = \text{MaxCor},
\end{align*}
\]

where \( \mu, \sigma \) and \( \rho \) are the mean, the standard deviation and the correlation coefficient, respectively. Now assume we randomly permute \( Y_1 \) as in Figure 7.1, \( X_1 \) and \( Y_1 \), hence, become uncorrelated and \( \rho_{X_1,Y_1} \) reduces from \( \text{MaxCor} \) to approximately \( 0 \). The correlation \( \rho_{X_2,Y_2} \) between \( X_2 \) and \( Y_2 \) is still \( \text{MaxCor} \) while the correlation between \( X \) and \( Y \) is reduced to

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

\(^4\)In statistics, when the size of a sample is large enough, the sample can be used to estimate statistical properties of its population such as mean, standard deviation, etc. In other words, these statistical properties of the sample are approximately equal to those of its population. In our study, each trace contains up to ten thousands of jobs, and hence the statistical properties of the sample are accurate estimators for the properties of the population.
The above proof concludes that our procedure can adjust accurately the degree of correlation on demand by specifying the requested degree via $d_j$.

7.2 Realistic Simulation Methodology

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

7.2.1 Applied Workload Model

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

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

7.2.2 Scheduling Policies

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

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

7.3.1 Performance Impact of Correlation

For this experiment, we create 11 patterns of workload, each corresponds to 11 different degrees of correlation and denoted by $W_j, j = 1, \ldots, 11$. At first, we apply the model in [67] on the trace HPC to generate a synthetic runtime process and a synthetic parallelism process. Since we do not need the presence of locality in this experiment, we randomly shuffle the synthetic runtime process so that it loses the locality structure. Then, we call the procedure Control Correlation with the 2 synthetic processes and a list of parameters \{d_j\} = \{-1, -0.8, -0.6, -0.4, -0.2, 0, +0.2, +0.4, +0.6, +0.8, +1\} as inputs to achieve 11 new synthetic parallelism processes, which associate with the shuffled runtime process to obtain 11 couples of runtime and parallelism processes with correlation ranging from -1 to +1. From these couples, we form 11 complete synthetic workloads $W_j, j = 1, \ldots, 11$ by using the Poisson distribution to
generate 11 job arrival processes that are aggregated with the couples. The Poisson parameter $\lambda$ is calculated such that all workloads produce the same load on the same system. Note that, all 11 workloads have the same runtime process and, therefore, the locality property will not affect their performance. We repeat this workload generation 20 times, so for each degree of correlation we have 20 different synthetic workloads, which results in 220 workloads used in this experiment.

![Graphs showing performance impact for different algorithms](image)

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

Table 7.1 describes the correlations of the generated synthetic workloads, calculated by Spearman’s approach. As we can observe, the results demonstrate that the procedure *Control_Correlation* works well since the degree of correlation can be efficiently controlled as shown in Section 7.1. Another noticeable point is that we only change the structure of runtimes (locality) and parallelisms (correlation) but not their
values. Therefore, all the synthetic workloads are representative with respect to the marginal distributions of runtimes and parallelisms since the applied model [67] can fit the marginal distributions well.

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

<table>
<thead>
<tr>
<th>Workload</th>
<th>( W_1 )</th>
<th>( W_2 )</th>
<th>( W_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation</td>
<td>(-0.957 \pm 0.001)</td>
<td>(-0.766 \pm 0.001)</td>
<td>(-0.576 \pm 0.003)</td>
</tr>
<tr>
<td>( W_4 )</td>
<td>(-0.382 \pm 0.002)</td>
<td>(-0.191 \pm 0.004)</td>
<td>(-0.001 \pm 0.004)</td>
</tr>
<tr>
<td>( W_5 )</td>
<td>(+0.384 \pm 0.003)</td>
<td>(+0.574 \pm 0.002)</td>
<td>(+0.766 \pm 0.002)</td>
</tr>
</tbody>
</table>

All of 220 synthetic workloads are run on the same simulated parallel system with 240 processors (the real system size of the cluster HPC). Figure 7.3 shows the performance impact of correlation on schedulers. We can see that for all cases, a larger degree of correlation leads to a performance degradation (larger slowdown). The reason is that when the correlation increases, longer runtimes tend to associate with larger numbers of processors and therefore, short jobs have to wait longer once processors are allocated to long jobs and this results in poorer performance. Especially, it can be seen in Figure 7.3 that the scheduling performance decreases slowly when the correlation changes from \(-1\) to \(0\) but then rapidly decreases when the correlation becomes positive and reaches \(+1\). The reason of this difference lies in the fragmentation problem. With a negative correlation, the number of processors associated with a long job is not so large and hence, a short job has more chance to be allocated with enough processors for execution. In general, the fragmentation issue happens rarely. However, with a highly positive correlation, the fragmentation problem seems to occur regularly once a long job is in execution and occupies a large number of processors. In this case, the number of free processors is small and not enough for waiting jobs. This leads to a fragmentation and a low utilization of the system. As a consequence, when a positive correlation reaches \(+1\), the fragmentation issue becomes more serious and the scheduling performance rapidly decreases. Another notable point drawn from Figure 7.3 is that the fragmentation issue of Aggressive and Categorized Selective policies is considerably alleviated compared with that of FIFO and Conservative policies. We explain this result by the backfilling behaviour. Fragmentation occurs when free processors are not allocated to waiting jobs due to reasons like the number of free processors is not enough or it is enough for a waiting job but the job cannot leapfrog other jobs that are waiting before it in a queue. This situation happens with FIFO policy, making the fragmentation issue worse. It also happens
with Conservative because despite backfilling is supported, this policy only backfills a job if it does not delay any previous jobs in the waiting queue. On the contrary, Aggressive and Categorized Selective are able to do more job backfilling, hence reduce the fragmentation issue. Therefore, we conclude that backfilling is especially good for positive correlation workloads. Since real workloads exhibit correlations ranging from around $-0.3$ to $+0.5$, we recommend that this correlation range should be carefully considered in evaluating scheduling algorithms.

![Graph](image.png)

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

Next to the performance impact of correlation, we also investigate how this feature affects scheduling design. Figure 7.4 compares the scheduling performance of two scheduling algorithms, namely Categorized Selective and Uncategorized Selective, under different degrees of correlation. The result is interesting since Categorized Selective is better than Uncategorized Selective with a negative correlation but turns to be worse when the correlation becomes positive. The reason is that Uncategorized Selective uses the same threshold for all jobs so that if a job has waited long enough, it will be provided with a reservation. Therefore, despite of the correlation, all jobs only have to wait up to the threshold to get a reservation for execution. This explains the relatively similar slowdowns for Uncategorized Selective when the correlation ranges from $-1$ to $+1$. In contrast, Categorized Selective uses different thresholds for different categories of jobs. Hence, one category can affect the wait times of the other categories. This results in an increase of the slowdown when the correlation reaches $+1$. Consequently, Categorized Selective turns to be worse than
Uncategorized Selective with an increase of correlation though it can be better with a negative correlation. Definitely, there is a potential risk of inaccurate scheduling design if using workloads without or with wrongly modeled correlation.

### 7.3.2 Performance Impact of Locality

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

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

### 7.4 Discussions

Our experimental results in Section 7.3 provide several clues that can help to design efficient scheduling policies. For illustration, we describe briefly in this section our idea to exploit the correlation and the locality features for enhancing scheduling performance. We define 4 states for a parallel system, consisting of overload-positive,
Chapter 7. Performance Impact of Runtime and Parallelism on Scheduling

Figure 7.5: Performance impact of locality on parallel systems.
overload-negative, underload-positive and underload-negative. The states are mostly
self-explanatory by their names, i.e., each represents the status of the system: whether
the system is over/under-loaded and whether the workload running on the system
exhibits a positive/negative correlation. Then we develop an individual scheduling
policy for each state. For example, a backfilling policy would be a good choice for posi-
tive correlation workloads to alleviate the fragmentation issue as explained in Section
7.3.1. In particular, a backfilling scheduling algorithm that restrains the wait time of
a job to a certain threshold like Uncategorized Selective is relatively good for a posi-
tive correlation as shown in Figure 7.4. In addition, a non-backfilling algorithm that
well utilizes the locality feature can be used when the system is overloaded. Whenever
the system falls into a state, the corresponding scheduling policy of the state is
called. For our approach to work, it is important to predict the future state of the
system. It is easy to know if a system is under/over-loaded by considering the waiting
queue of the system and the status of computing nodes. An efficient predictor needs
to be developed for anticipating a state with negative/positive correlation. We tried
a relatively simple predictor, that is to estimate the correlation of one week by the
average of that over the previous 2 weeks. We divide each long-term trace in Table
3.4 into a series of consecutive weeks and evaluate this predictor. For a week, we
compare the estimated correlation with the real correlation of the week. If both give
the same positive/negative correlation, we count the week as one correctly predicted
time. Consequently, the accurateness in term of number of times that are correctly
predicted, is approximately 74% on average in the 12 traces in Table 3.4. Note that,
this is just a simple predictor and an advanced predictor should be developed. As a
whole, it is predictable for the future state of the system and, therefore, our idea is
feasible giving us a promising future work.

7.5 Summary

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