Improving the Execution of Groups of Simulations on a Cluster of Workstations and its Application to Storage Area Networks

À. Perles X. Molero A. Martí V. Santonja J.J. Serrano
Departament d’Informàtica de Sistemes i Computadors
Universitat Politècnica de València
Camí de Vera, s/n. 46022 València, Spain
Tel: (+34) 96 3879701 Fax: (+34) 963877579
email: aperles@disca.upv.es

Abstract

Parallel simulation methods can be used to reduce the execution time of simulations of complex systems. This approach is being used to improve the execution time of a storage area network (SAN) simulator designed in our department. From our experience in planning simulation experiments, we have realized that, in most cases, a simulation experiment (group of simulations) is executed while varying only one input variable, which usually corresponds to the input workload or a configuration model parameter.

In this paper we propose two methods to reduce the overall execution time of a simulation experiment using a cluster of workstations. The first method uses the first simulation in order to tune the rest of the remaining work to be done in the experiment. The second method, based in the first one, tries to minimize the negative influence of the initial transient period by chaining the simulations in the experiment. We show that these two methods noticeably decrease the overall execution time needed to run the simulations that compose the experiment.

1. Introduction

The discrete-event simulation techniques have demonstrated to be an indispensable tool for the analysis and the development of all technologies susceptible to be outlined using simulation models. However, many researches have suffered the huge computational resources needed and long response times executing simulation models moderately complex.

A widely used simulation technique is the replication method [6]. This method is based on the independence hypothesis of the random input variable of each replication and it seeks to calculate a confidence interval for the selected output variable. The sequential and classic replication method consists of running fixed size simulations and testing if they have been reached the wished confidence intervals. In this scheme, each replication starts in the initial empty state. If the replications are too short it will be necessary to start another simulation execution of a longer size. On the other hand, if they are too long, the results will be unnecessarily more accurate, wasting computer resources and increasing response time. The difficulty to predict the number of replications and their respective lengths (simulated time) has motivated the development of alternative methods that relax the independence hypothesis, at the same time they are easily implemented in a sequential computer. Some examples are both the batch-means and the regeneration methods, which add the advantage that only one initial transient period must be eliminated (for steady-state simulations).

Nowadays, a lot of research work is being spent in order to improve the response times of simulators by using distributed simulation techniques that can be implemented in a cluster of workstations. A main work line develops the possibility of decomposing the simulation model in sub-models that can be executed concurrently. The main problem of this approximation consists of maintaining the logical consistency of the event list [4,14]. The use of distributed simulation techniques is not an easy task and requires both a great decomposition effort and model tuning. Anyway, a deficient tuning process produces poor speed-ups.

An alternative to the distributed simulation consists of executing several copies of the simulation model in parallel. As in the case of distributed simulation, this parallelism is also well suited to be implemented in a cluster of workstations. This approach could be used for improving response times taking advantage of the “statistical parallelism”. Environments for the development of parallel and distributed applications on a cluster of workstations provide a new horizon to the statistical analysis of simulation output variables. Some authors [2,7,13] proposed tools that attempt to use a group

This work has been supported by the CICYT TAP99-0443-C05-02
of workstations in order to decrease response time by means of the parallel replication method and others.

The parallel simulation approach is being used to improve the execution time of a simulator of storage area networks (SAN), designed in our department, by means of the Clustersed Simulation Experimenter (CSX) tool [11]. This environment has been implemented to carry out massive parallel simulation experiments without having to be concerned in locating computational resources to carry it out. The main benefits obtained when applying this tool to our SAN simulator have been presented in [12]. Our approximation basically consists of executing variable-sized independent replications of the simulation model in parallel. Moreover, from our experience in planning simulation experiments, we have realized that, in most cases, a simulation experiment (group of simulations) is executed while varying only one input variable, that typically corresponds to the input workload or a configuration model parameter.

In this paper we propose two methods to reduce the overall execution time of a simulation experiment in a cluster of workstations by taking into account that simulations belonging to the experiment are executed while varying only the input workload. The first method is based on the fact that the critical factor for obtaining the best speed-up for the variable-sized parallel-replication scheme is the selection of the number of replications. Then, we use the first replication in order to tune the rest of the work. On the other hand, the second method, based in the first one, tries to minimize the negative influence of the initial transient period when use the classical replication technique. As suggested in the literature, this effect can be reduced if the model departs near the steady state. Since all the simulations in a experiment differ only in the input workload, once a simulation has finished, we can use its final steady state as the initial (non-empty) state in order to run the following simulation.

The rest of the paper is organized as follows. Section 2 gives some highlights about the parallel simulation environment we have employed to carry out the experiments. The SAN simulation model we have used in this study is briefly detailed in Section 3. The two proposed simulation methods are explained in Section 4. The analysis of simulation results is carried out in Section 5. Finally, Section 6 summarizes the conclusions provided by our study.

2. Parallel simulation environment: CSX

The main purpose of the CSX (Clustered Simulation Experimenter) tool [11] is to use a lot of idle and heterogeneous workstations at university laboratories and research centers to run concurrent simulations. Simulations executed using CSX are discrete-event simulation models that, basically, will be run using replication techniques, and will be monitored in order to extract statistical information of the selected output variables.

The CSX design allows its application to any commercial or public simulator, avoiding the apprenticeship of a new simulation language. It provides a general form of monitoring and controlling simulations and it has been initially applied to the SMPL simulation language [8]. The monitorization of the model only requires minimal changes in the original program code, and it is mainly based on a special event injection in the simulator event queue.

This tool has been outlined as a distributed application working under PVM [5] that lets Unix and Windows heterogeneous computer incorporation to a parallel virtual machine. In this work, this environment has been successfully applied to the CSIM simulation language [1].

When using the CSX tool, a parallel simulation can be done by spawning N replications of the instrumented program. Each replication uses a different random stream, and the output variables of each replication are used to obtain a confidence interval.

3. Our SAN simulation model

Storage area networks (SANs) [3] are an emerging data communications platform that interconnects servers and storage devices to create a pool of storage that users can access directly. This networking approach reports benefits such as computer clustering, topological flexibility, fault tolerance, high availability, and remote management.

In order to evaluate the performance of these systems, a very flexible and easy to use SAN simulator has been developed [9]. This simulator has been written using CSIM libraries [1]. A CSIM program models a system as a collection of CSIM processes that interact with each other by using internal structures to CSIM. The simulator, which nowadays is about 12,500 lines long, has been written in ANSI C code in order to enable system portability. In fact, we have run it in both Unix and Windows systems.

As this tool accurately models a SAN focusing on its internal design, it produces very computationally expensive simulation programs. The simulator uses the built-in batch-means simulation technique to statistically test output variables. However, obtaining accurate results appears to be computationally expensive.

Figure 1 shows a typical result obtained by using this SAN simulation model, where each represented point corresponds to a sim() simulation process with a different value of the input workload. This workload is computed as the total amount of traffic injected into the interconnection network and is measured in bytes per clock cycle. As can be observed, the system will be in a operational state while delivered traffic equals the injected traffic (linear region).
In this study we have considered a SAN with the *floors* topology [10] shown in Figure 2. In this topology, which is used in building environments, most connections are between switches and devices in each of the floors. The structure on each floor, shown in Figure 2(b), consists of a star with center switches and arm switches. This structure is replicated as many times as floors are included in the storage area network. Center switches are specifically devoted for connecting switches at different floors. Servers and storage devices are attached to the SAN by means of the arm switches. This topology has several connections to provide a degree of fault tolerance. Our case study is a 5-floor topology where each floor has 2 center switches. Each center switch is connected to all the center switches at the adjacent floors. Four arm switches provide connectivity at each floor. Each of the arm switches is connected to a single server and 5 disks. Thus, 20 servers and 100 disks compose our storage area network.

![Figure 2. SAN topology considered in our study](image)

### 4. Simulation methods

This section describes the different simulation methods compared in our study. The two methods called A and B corresponds to the classical approaches for both sequential and parallel simulations. Our two proposed methods are then explained and will be called C and D. We will assume that the simulation model must be solved for *w* different values of the input workload, and the simulations are executed on a cluster composed of *n* workstations, where *w* ≤ *n*.

#### 4.1 Method A: sequential execution

This method corresponds to the classical sequential approach. The simulation model is executed on one workstation for each value of the workload (see Figure 3). In each simulation, the batch means technique is used to obtain the confidence interval for the mean of the selected output variable. As can be deduced, all workload points are solved by means of different executions of the simulation model. These independent executions can be carried out in parallel, each one on a different machine in the cluster. Each simulation starts in an empty initial state.

![Figure 3 Method A: independent simulations for each workload point](image)

If we start the simulations of the experiment at the same time, then the amount of time *T* that the experiment will take can be computed as the execution time of the longest execution, that is

\[
T = \max_{i=1}^{w} t_i
\]

where *w* is the number of workload points, and *t* is the time needed to simulate the workload point *i*.

#### 4.2 Method B: parallel execution

In this method the simulation of each workload point is carried out by as many independent replications executed in parallel as workstations in the cluster. The mean value of the output variable is computed by combining the results obtained in all the independent replications. As it has been pointed out, each parallel replication contains a transient period that contributes to the global transient period.

All simulations are started in an empty initial state. The execution of the simulation model for each workload point is carried out in a sequential manner, as depicted in Figure 4.
Transient periods started in the initial empty state

Global transient period

Workload point 1

Global steady state period

Workload point 2

workstations

n workstations

Global transient period

Workload point w

Global steady state period

n workstations

Figure 4 Method B: parallel independent replications for each workload point

The total amount of time $T$ that the simulation experiment will take can be computed as the sum of the execution times invested in the simulation of all the workload points, that is,

$$T = \sum_{i=1}^{w} t_i$$

where $w$ is the number of workload points, and $t_i$ is the time needed to simulate the workload point $i$.

4.3 Method C: parallel execution tuning the number of independent replications

In this method, the simulation of the first workload point is used to tune the appropriate number of independent replications that should be used in the remaining simulation points. This tuning process will take into account the relationship among the workstations available in the cluster, their processing power, and the particular behavior of the simulation model.

Figure 5. Method C: first simulated workload point is used to tune the rest of the experiment

Figure 5 graphically shows the application of this method. As can be seen, the first point is executed by using all the workstations in the cluster, each one executing an independent replication. Once the first point has been simulated, the information reported can be used to tune the execution of the remaining simulations. In this case, the second workload point is executed on $n_1$ workstations, and the third one is executed on $n_2$ workstations, with $n=n_1+n_2$. It is assumed that the two subsets of workstations remain the same along the subsequent simulations. As in the previous two methods, all the replications start in an initial empty state.

The total amount of time $T$ that the simulation experiment will take can be computed by adding the execution time of the first simulation and the longest execution time of one of the subsets of the workload points, that is,

$$T = t_1 + \max \left\{ \sum_{i=2}^{w} t_i, \sum_{j=2}^{w} t_j \right\}$$

where $w$ is the number of workload points, and $t_i$ represents the time needed to simulate the workload point $i$. As will be seen below, the distribution of the simulations is a critical factor in the reduction of the simulation time.

4.4 Method D: parallel execution tuning the number of independent replications and avoiding the initial empty state

The main objective of this method is to avoid the initial empty state in which start all the simulations described in the previous scheme. The literature suggests that it is possible to reduce the effect of the initial transient period (warm up) by starting the simulation in a state near the steady state. To do so, a small variation can be applied to the previous approach in the sense that, once the simulation of the first workload point has been finished, all the remaining points could start in the steady state just achieved. As in method C, the simulation of the first workload point is used to tune the appropriate number of independent replications that should be used in the remaining simulation points. Moreover, once the simulation process has been tuned, the subsequent independent replications will be reconfigured to continue with the execution of the following workload points, but these replications will not start in the initial empty state.

Figure 6 graphically shows the application of this method. The first point is executed by using all the workstations in the cluster, each one executing an independent replication. Once the first point has been simulated, it can be used to tune the rest of simulations. In this case, the second workload point is executed on $n_1$ workstations, and the third one is executed on $n_2$ workstations, with $n=n_1+n_2$. However, in contrast to the previous methods, all the replications in the second and subsequent workload points start in an initial non-empty state.
The total amount of time \( T \) that the simulation experiment will take can be computed as in the previous method by adding the execution time of the first simulation and the longest execution time of one of the subsets of the workload points, that is,

\[
T = t_1 + \max \left\{ \sum_{j=2}^{w} t_j, \sum_{j=\frac{w+1}{2}}^{w} t_j \right\}
\]

where \( w \) is the number of workload points, and \( t_i \) represents the time needed to simulate the workload point \( i \). However, it is expected that in that case \( t_i \) could be decreased by the effect of removing the initial empty state.

5. Experimental results

This section compares the total amount of time \( T \) taken by a simulation experiment when using the different simulation methods described in the previous section. A cluster composed of 10 PCs AMD K6-2/350 MHz with 128 MB RAM and running SuSE 6.1 Linux has been utilized in order to run our experiments \((n=10)\). The considered output variable is the "I/O mean response time" experienced by the I/O operations. This time includes the network latency, the disk access time, and the queuing time. We have considered 7 different levels \((w=7)\) for the input workload parameter of the simulation model. These workload levels are 8.5, 11, 13.5, 16, 18.5, 21, and 23.6, all measured in bytes/cycle, where one clock cycle is equivalent to 6.5 nanoseconds [9].

The original SAN simulator program code has been modified and linked with CSX libraries in order to be monitored in the parallel environment. When CSX receives a sample of statistical information of any replication, it computes the confidence interval for the user-selected output variable. The simulation will end if the confidence interval satisfies the user-specified end conditions. We have considered a 95% confidence interval for the mean. The stopping simulation criterion is achieved when the relation between the mean and the half-width of the confidence interval is lower than 5% and this error decreases monotonically.

5.1 Method A: sequential execution

This experiment has been carried out by using the CSIM built-in batch means method. Due to the fact that the source code of CSIM is not available and the specific batch means is not accurately described, we have considered that this method is a black box that offers two outputs: a confidence interval and a logical variable indicating if the simulation has converged.

Table 1 summarizes the obtained results for each workload level. Mean and half-width confidence interval, and the execution time of each simulation are shown. As can be appreciated, the execution time increases as workload level increases. We think that this effect is due to the fact that CSIM increments the number of collected batches when no convergence has been achieved in a previous phase. For the highest workload level, the CSIM batch means method has not converged although the simulation time has been 10 hours.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Mean</th>
<th>H</th>
<th>Execution time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>4281.9</td>
<td>61.1</td>
<td>50</td>
</tr>
<tr>
<td>11</td>
<td>4875.8</td>
<td>106.2</td>
<td>52</td>
</tr>
<tr>
<td>13.5</td>
<td>5528.2</td>
<td>117.6</td>
<td>51</td>
</tr>
<tr>
<td>16</td>
<td>6552.1</td>
<td>154.5</td>
<td>104</td>
</tr>
<tr>
<td>18.5</td>
<td>7649.2</td>
<td>180.7</td>
<td>199</td>
</tr>
<tr>
<td>21</td>
<td>9255.0</td>
<td>177.6</td>
<td>109</td>
</tr>
<tr>
<td>23.5</td>
<td>11583.4</td>
<td>not available</td>
<td>600</td>
</tr>
</tbody>
</table>

Table 1. Mean and half-width interval for the "I/O response time" output variable for method A

From data in Table 1, the overall time needed to get the results of the experiment is:

\[
T = \max_{j=1}^{w} t_j = \max\{50,52,51,104,199,109,600\} = 600 \text{ minutes}
\]

5.2 Method B: parallel execution

In order to carry out this experiment, 10 independent replications (different input random streams) are spawned on the cluster, each one in one workstation. Statistical information related to the evolution of the objective variable is sent periodically to CSX in order to compute the grand mean and its half-width confidence interval.

The effect of the initial warm-up has been analyzed before carrying out the definitive experiment. Figure 7 shows the evolution of the mean and the relative error for 10 replications when applying a reset of the statistics in the point that seems more appropriate. As can be seen, the
reset produces that the mean reaches the steady-state value earlier, the tendency of the relative error stays the same that in the case in which the reset is not used. Thus, for this particular model, a restart of statistics after the warm-up period does not contribute to a significant convergence speed.

![Graph](image)

Figure 7. Mean and relative error evolution. 10 replications and workload level equal to 16

Table 2 summarizes the obtained results for each workload level.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Mean</th>
<th>H</th>
<th>Execution time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>4253.6</td>
<td>92.9</td>
<td>3.1</td>
</tr>
<tr>
<td>11</td>
<td>4840.0</td>
<td>105.2</td>
<td>4.0</td>
</tr>
<tr>
<td>13.5</td>
<td>5508.6</td>
<td>96.9</td>
<td>5.9</td>
</tr>
<tr>
<td>16</td>
<td>6348.1</td>
<td>72.0</td>
<td>9.5</td>
</tr>
<tr>
<td>18.5</td>
<td>7484.7</td>
<td>95.5</td>
<td>14.7</td>
</tr>
<tr>
<td>21</td>
<td>9081.0</td>
<td>162.8</td>
<td>27.9</td>
</tr>
<tr>
<td>23.5</td>
<td>11328.5</td>
<td>289.8</td>
<td>45.7</td>
</tr>
</tbody>
</table>

Table 2. Mean and half-width interval for the "I/O response time" output variable for method B

From data in Table 2, the overall time needed to get the results of the experiment is:

\[ T = \sum_{i=1}^{w} t_i = 3.1 + 4.0 + 5.9 + 9.5 + 14.7 + 27.9 + 45.7 = 110 \text{ minutes} \]

5.3 Method C: parallel execution tuning the number of independent replications

In this case the 10 available workstations are used to execute one replication on each machine for the first workload point. The simulation will be allowed to run enough to discover which is the adequate number of replications based on the model behavior and the processing power of the machines. The basic idea is very simple: by means of the execution of 10 replications during a period of time long enough, we can calculate how much time will be invested to reach the convergence when 9, 8, 7... 2 replications are used. This time may be a good indicator about the model behavior when simulating the rest of workload levels.

Figure 8 shows the computed simulation end time for different number of replications when a workload level of 8.5 is used. In order to select the ideal number of replications, the more feasible possibilities are 3, 5 or 10 replications. The use of only 3 replications may be an unsafe election from the point of view of the mean coverage, whereas using 10 replications will waste a lot of computer resources without providing an important performance compared with 5 replications.

![Graph](image)

Figure 8. Execution time for different numbers of replications and workload level 8.5

Taking into account the cluster of workstations used to run our experiment, it seems that the most appropriate number of replications is 5. Then, the workload levels 11, 13.5 and 16 could be executed in a first group of 5 machines, and the levels 18.5, 21 and 23.5 on the remaining group. Table 3 summarizes the obtained results for each workload level.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Mean</th>
<th>H</th>
<th>Execution time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>4289.4</td>
<td>106.5</td>
<td>3.9 (6.0)</td>
</tr>
<tr>
<td>11</td>
<td>4792.1</td>
<td>113.3</td>
<td>5.1</td>
</tr>
<tr>
<td>13.5</td>
<td>5569.3</td>
<td>124.3</td>
<td>7.8</td>
</tr>
<tr>
<td>16</td>
<td>6432.6</td>
<td>79.1</td>
<td>13.1</td>
</tr>
<tr>
<td>18.5</td>
<td>7527.3</td>
<td>100.9</td>
<td>21.6</td>
</tr>
<tr>
<td>21</td>
<td>9109.7</td>
<td>265.4</td>
<td>32.5</td>
</tr>
<tr>
<td>23.5</td>
<td>11251.2</td>
<td>247.6</td>
<td>48.8</td>
</tr>
</tbody>
</table>

Table 3. Mean and half-width interval for the "I/O response time" output variable for method C

From data in Table 3, the overall time needed to get the results of the experiment is:

\[ T = t_1 + \max \left( \sum_{j=2}^{w+1} t_j, \sum_{j=2}^{w+1} t_j \right) = \]

\[ T = t_1 + \max \left( \sum_{j=2}^{w+1} t_j, \sum_{j=2}^{w+1} t_j \right) = \]
The analysis of the previous expression shows us that the execution time of the experiment has been penalized due to the fact that one of the groups of 5 machines has carried out the most expensive simulations. Therefore it is advisable to look for on-line procedures in order to sort the simulations of different workload levels. For example, if we schedule the next workload point to the group of workstations that finishes a particular point, the experiment will get:

\[ T = 6.0 + \max\{5.1 + 7.8 + 13.1 + 21.6 + 32.5 + 48.8\} = 84.2 \text{ minutes} \]

We will call this method C2 in order to differentiate from the previous one, that we will call C1.

On the other hand, if we alternatively schedule the next and the last point to the group of workstations that finishes a particular point, then the experiment will get:

\[ T = 6.0 + \max\{5.1 + 7.8 + 13.1 + 21.6 + 32.5\} = 86.1 \text{ minutes} \]

We could do a best share of the work, for example, by executing the workload levels 11, 13.5 and 23.5 on a group of machines, and levels 16, 18.5 and 21 on the other group. In this case, the overall time provided is:

\[ T = 6.0 + \max\{5.1 + 7.8 + 13.1 + 21.6 + 32.5\} = 73 \text{ minutes} \]

However, this ideal result can not be obtained on-line.

5.4 Method D: parallel execution tuning the number of independent replications and avoiding the initial empty state

The first simulated point is used, as in the previous method, to tune the appropriate number of replications. In this case, based on the precedent results, the experiment will be executed in this way: workload levels 11, 18.5 and 21 on a group of machines, and levels 13.5, 16 y 23.5 on the other group.

Figures 9 and 10 show the evolution of the mean and the relative error for workload levels 8.5, 11, 18.5 and 21, and levels 8.5, 13.5, 16 and 23.5, respectively. Also, Table 4 summarizes the obtained results for all the considered workload levels.

![Figure 9](image9.png)

Figure 9. Mean and relative error evolution for workload levels 8.5, 11, 18.5 and 21

![Figure 10](image10.png)

Figure 10. Mean and relative error evolution for workload levels 8.5, 13.5, 16 and 23.5

<table>
<thead>
<tr>
<th>Workload</th>
<th>Mean</th>
<th>H</th>
<th>Execution time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>4289.4</td>
<td>106.5</td>
<td>3.9 (6.0)</td>
</tr>
<tr>
<td>11</td>
<td>4694.7</td>
<td>112.8</td>
<td>5.1</td>
</tr>
<tr>
<td>13.5</td>
<td>5367.4</td>
<td>156.5</td>
<td>7.8</td>
</tr>
<tr>
<td>16</td>
<td>6378.3</td>
<td>79.2</td>
<td>4.2</td>
</tr>
<tr>
<td>18.5</td>
<td>7494.7</td>
<td>105.1</td>
<td>12.1</td>
</tr>
<tr>
<td>21</td>
<td>9003.9</td>
<td>231.1</td>
<td>30.2</td>
</tr>
<tr>
<td>23.5</td>
<td>11729.4</td>
<td>282.3</td>
<td>44.6</td>
</tr>
</tbody>
</table>

Table 4. Mean and half-width interval for the "I/O response time" output variable for method D ("cold reset" and chained experiments)

From data in Table 4, the overall time needed to get the results of the experiment is:

\[ T = 6.0 + \max\{5.1 + 12.1 + 30.2, 7.8 + 4.2 + 44.6\} = 62.6 \text{ minutes} \]

5.5 Summary results

All tested methods offer the same result for the "I/O response time" output variables from a statistical point of view because all intervals overlap for each input workload level.
Figure 11 shows the overall execution time for the different methods. As can be seen, parallel execution highly improves execution time.

As can be seen, methods B and C1 seem to provide a similar performance, but as it has been pointed out, this is due to the effect of static planning to run the different simulations.

Methods C2 and C3 are two dynamic approaches of the initial C method that provide a better use of the computational resources. Therefore, we can lose the benefits of the tune phase if we do not provide a dynamic dispatch of next simulations. The best results of this method will be obtained with this dynamic dispatch of work, and it is expected that a greater number of levels will benefit the results.

Also, it is possible to chain simulations in order to reduce the effect of the warm-up period. This approximation has improved the total execution time.

### References


<table>
<thead>
<tr>
<th>Method</th>
<th>Execution time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>600</td>
</tr>
<tr>
<td>B</td>
<td>110</td>
</tr>
<tr>
<td>C1</td>
<td>108</td>
</tr>
<tr>
<td>C2</td>
<td>84.2</td>
</tr>
<tr>
<td>C3</td>
<td>86.1</td>
</tr>
<tr>
<td>D</td>
<td>62.6</td>
</tr>
</tbody>
</table>

Figure 11. Overall execution time for the different simulation methods

Finally, method D provides a slightly better performance than the dynamic version of method C. This better performance can also benefit of an increment and dynamic dispatch of the simulation levels. If we look to the result tables it seems that this method provides an underestimation of the expected value of the "I/O response" variable (except workload level 23.5).

Table 5 shows the speed-up of the parallel methods with respect to method B. As it can be appreciated, method D obtains the better speedup, whereas methods C2 and C3 are more efficient than C1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C1</td>
<td>1.02</td>
</tr>
<tr>
<td>C2</td>
<td>1.31</td>
</tr>
<tr>
<td>C3</td>
<td>1.28</td>
</tr>
<tr>
<td>D</td>
<td>1.76</td>
</tr>
</tbody>
</table>

Table 5. Speedup with respect to the B method

### 6. Conclusions

In this paper we have proposed two methods that reduce the overall execution time needed to carry out a simulation experiment by using a cluster of homogeneous workstations. These methods take into account that simulations belonging to the experiment are executed while varying only the input workload. We have showed that it is possible to decrease the overall execution time of the experiment by using the first parallel replication in order to get later a better performance for the subsequent simulations in the experiment.