# Dynamic Load Balancing and Job Replication in a Global-Scale Grid Environment: a Comparison

Menno Dobber[†], Rob van der Mei[†§], and Ger Koole[†]

[†] Vrije Universiteit, Faculty of Sciences, Amsterdam, The Netherlands

[§] Center for Mathematics and Computer Science, Amsterdam, The Netherlands

Global-scale grids provide a massive source of processing power, providing the means to support processor intensive parallel applications. The strong burstiness and unpredictability of the available processing and network resources raise the strong need to make applications robust against the dynamics of grid environments. The two main techniques that are most suitable to cope with the dynamic nature of the grid are Dynamic Load Balancing (DLB) and job replication (JR). In this paper, we analyze and compare the effectiveness of these two approaches by means of trace-driven simulations. We observe that there exists an easy-to-measure statistic $Y$, and a corresponding threshold value $Y^*$, such that DLB consistently outperforms JR when $Y > Y^*$, whereas the reverse is true for $Y < Y^*$. Based on this observation, we propose a simple and easy-to-implement approach, throughout referred to as the DLB/JR method, that can make dynamic decisions about whether to use DLB or JR. Extensive simulations based on a large set of real data monitored in a global-scale grid show that our DLB/JR method consistently performs at least as good as both DLB and JR in all circumstances, which makes our DLB/JR method highly robust against the unpredictable nature of global-scale grids.

**Keywords:** Grid computing, dynamic load balancing, job replication, performance

## 1 Introduction

Variations in the available resources (e.g., computing power, bandwidth) may have a dramatic impact on the running times of parallel applications. Over the years, much research has been done on this subject in grid computing. Generally, two methods for parallel applications have been developed to deal with those fluctuations in processor speeds on the nodes: Dynamic Load Balancing (DLB) (e.g., [3, 5, 8, 9, 18, 22, 23]),

and job replication (JR) [2, 4, 6, 7, 14, 17, 21, 16, 23]. DLB adapts the load on the different processors in proportion to the expected processor speeds. JR makes a given number of copies of each job, sends the copies and the original job to different processors, and waits until the first replication is finished. A comparison of the performance of those two methods on a heterogeneous globally distributed grid environment has - to the best of the author's knowledge - never been performed.

Recently, a variety of grid test-beds have been developed (e.g., Planetlab [1]). This enables us to perform comprehensive measurements of realistic job times to investigate how well certain implementations of grid applications perform in practice for a wide range of different experimental setups. In this paper, we provide extensive trace-driven simulation experiments of dynamic load balancing (DLB) and job replication (JR) as two implementation concepts to deal with the ever-changing environment on widespread grid nodes. Moreover, we introduce a new selection method that dynamically selects the best implementation and show its effectiveness in global-scale grid environments.

In general, two types of investigations have been applied to accurately verify whether certain algorithms or implementations are effective: (1) trace-driven simulation, and (2) real implementation. On the one hand, the first type is (1) often easier to implement than a real implementation, because no advanced communication implementations are necessary, and (2) less clock time is needed to test different experimental setups. Consequently, trace-driven simulations need a shorter time period in which more situations can be analyzed. For example, previously done research [9] effectively shows the performance gain of a real implementation of DLB in a real grid environment. As many as 60 days of parallel-implementation runs were necessary to derive a reliable estimation of the performance improvement (speedup) of DLB compared to Equal Load Balancing (ELB) on four processors. Trace-driven simulations (e.g., [8, 23]) would take less time and more extensive analyses can be performed. On the other hand, trace-driven simulations require detailed knowledge about the processes. To this end, in this paper we program a real implementation of DLB to acquire more knowledge about the durations of the different processes within an application which is based on DLB.

The computations and commmunications structure of many parallel applications can be described by the Bulk Synchronous Parallel (BSP) model (cf. [19]). The relevance of the BSP model lies in the fact that it has the important property that the problems can be divided into sub-problems, each of which can be solved or executed in roughly the same way. As such, in the absence of any prior knowledge about the processor speeds and link rates in large-scale grid environments, the BSP model can be seen as a default means to parallelize computationally intensive applications. The BSP model includes the structure of Single-Program-Multiple-

Data (SPMD) [13], which is a classification of the structure of many parallel implementations. Currently, not many of the BSP type of applications are able to run in a grid environment, due to the fact that they cannot deal with the ever-changing environment. Especially, the synchronization in BSP programs causes inefficiency: one late job can delay the whole process. This raises the need for methods that make the BSP applications robust against changes in the grid environment.

In this paper, we analyze and compare the effectiveness of ELB, DLB and JR, using trace-driven simulations based on real data gathered in a global-scale grid test bed, called Planetlab [1]. The results show that both DLB and JR strongly outperform the default ELB, which is widely deployed in grid environments today. Further, an extensive comparison between DLB and JR reveals that in some circumstances JR performance better than DLB, but in other circumstances DLB is preferable. Given the strong unpredictability of the circumstances in the grid environment, this observation makes it difficult to assess the relative effectiveness of DLB or JR. Nonetheless, in-depth analysis shows that we can identify an easy-to-measure statistic $Y$ and a corresponding threshold value $Y^*$ such that DLB consistently outperforms JR for $Y > Y^*$, whereas JR consistently performs better for $Y < Y^*$. This observation naturally leads to a simple and easy-to-implement approach that can make on-the-fly decisions about whether to use to DLB or JR. Extensive simulation experiments show that this DLB/JR method always performs at least as good as both DLB and JR in all circumstances. As such, the DLB/JR methode presented in this paper provides a highly effective means to make parallel applications robust in large-scale grid environments.

This paper is organized as follows. In Section 2, we introduce the concept of Bulk Synchronous Processing. Moreover, we describe two different implementation types to deal with fluctuations in grid environments: dynamic load balancing (DLB), and job replication (JR). In Section 3, we describe the details of data-collection procedure and the implementation details of the trace-driven simulations of DLB, JR, and the selection method. Next, in Section 4, we show the results of the extensive experiments. Finally, in Section 5, we formulate the conclusions.

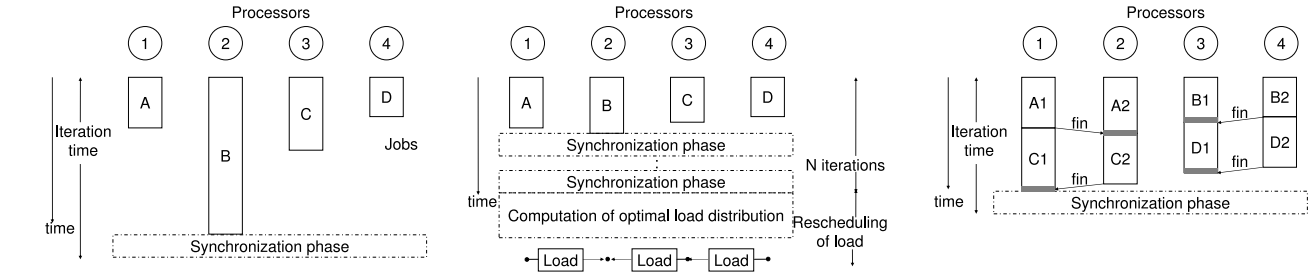
# 2 Preliminaries

In section 2.1 we briefly describe the concept of the Bulk Synchronous Parallel (BSP) model. Then, in Section 2.2 we describe the implementation details of DLB and JR.

## 2.1 Bulk Synchronous Processing

BSP parallel programs have the property that the problem can be divided into sub-problems or jobs, each of which can be solved or executed in roughly the same way. Each run consists of $I$ iterations of $P$ jobs which

are distributed on $P$ processors: each processor receives one job per iteration. Further, every run contains $I$ synchronization moments: after computing the jobs, all the processors send their data and wait for eachothers data before the next iteration starts. In general, the run time equals the sum of the individual iteration times. Figure 1(a) presents the situation for one iteration of a BSP run in a grid environment. The figure shows that each processor receives a job and the iteration time equals the maximum of the individual job times plus the synchronization time. Equal load balancing (ELB) assumes no prior knowledge of processor speeds of the nodes, and consequently balances the load equally among the different nodes. The standard BSP program is implemented according to the ELB principle.




(a) **ELB** (b) **DLB** (c) **2-JR**

Figure 1: The different types of implementations on four processors: ELB, DLB, and 2-JR

## 2.2 Load Balancing and Job Replication

In this section, we briefly discuss the two main methods to cope with the dynamics of the grid environment: DLB and JR.

### 2.2.1 Dynamic Load Balancing

DLB starts with the execution of an iteration, which does not differ from the common BSP program explained above. However, at the end of each iteration the processors predict their processing speed for the next iteration. We select one processor to be the DLB scheduler. After every $N$ iterations the processors send their prediction to this scheduler. Subsequently, this processor calculates the "optimal" load distribution given those predictions and sends relevant information to each processor. The load distribution is optimal when all processors finish their calculation exactly at the same time. Therefore, it is "optimal" when the load assigned to each processor is proportional to its predicted processor speed. Finally, all processors redistribute the load. Figure 1(b) provides an overview of the different steps within a DLB implementation on 4 processors. The effectiveness of DLB partly relies on the dividing possibilities of the load.

Load balancing at every single iteration is rarely a good strategy. On the one hand, the running time of a parallel application directly depends on the overhead of DLB, and therefore it is better to increase the number of iterations between two load balancing steps. On the other hand, less load balancing leads to an imbalance of the load for the processors for sustained periods of time, due to significant changes in processing speeds. In [8] we present the theoretical speedups in running times when using DLB compared to ELB, given that the application re-balances the load every $N$ iterations, but without taking into account the overhead. Based on those speedups and the load balancing overhead addressed above, a suitable value of $N$ was found to be $2.5P$ iterations, for $P > 1$. The effectiveness of DLB strongly relies on the accuracy of the predictions. Previous research [10] has shown that the Dynamic Exponential Smoothing (DES) predictor accurately predicts the job times on shared processors. For that reason, the DES predictor has been implemented in the DLB-simulation implementation of this paper.

#### 2.2.2 Job Replication

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

Figure 1(c) shows the situation for a 2-JR run on four processors. As can be seen in the figure, each job and its copy are distributed to $R = 2$ processors and during one iteration each processor receives $R = 2$ jobs. Processor one finished as first job A and sends a 'finalize' message to processor two. Sending the message over the internet takes some time and, therefore, it takes a while before the other processors start the next job. Each job-type time, which is the duration of a specific job type (the original and its copies), equals the minimum of all its job times plus a possible send time. An individual processor time of one iteration equals the sum of the job-type times which were sent to that processor and the send times of the 'kill'-messages. Finally, the iteration time of all the processors corresponds to the sum of the synchronization time and the maximum of all processor times.

### 2.2.3 Selection method

Next, we introduce the concept of a method that selects between the above two different types of implementations. This method has the aim to select dynamically the optimal implementation type. To this end, it measures a statistic $Y$ and defines a threshold $Y^*$ during the run. After each $I_p$ iterations, the method gives a preference for a given type of implementation, based on a comparison of $Y$ with the threshold value $Y^*$. In this method, the processor that redistributes the load in DLB is moreover the processor that decides whether JR or DLB is used. When the method decides that a switch to the other type of implementation is necessary, the steps to be taken are the same as in the DLB rescheduling phase: (1) the nodes send their prediction to the scheduler, (2) the scheduler computes the optimal load distribution, and (3) the nodes redistribute their load.

## 3 Experimental setup

In this section, we describe the grid test-bed, the data-collection procedure, the simulation details of DLB and JR, the development of the selection method and, finally, we introduce the method.

### 3.1 Data-collection procedure

In order to perform extensive investigations with the different types of implementations, in total 130 runs have been performed on 22 different heterogeneous processors of Planetlab [1]. The processors are globally distributed, shared with others, and the capacities of the processors are unknown. Each run consists of 2000 consecutive and identical jobs (or computations) and generates a dataset of 2000 job times. We constructed the jobs such that on a completely available Pentium 4, 3.0GHz processor, the computations in the jobs would take 10000 ms. The time between successive runs that are performed on the same processor ranges from one day to one month. We notice that the more time between the runs, the more difference between the characteristics of the job times of those runs. This is due to the fact that due to the highly random nature of global-scale grid environments, the dependencies of the characteristics of the system will be highly dependent over short timescales (e.g., during sudden traffic bursts, causing strong correlations), whereas these dependencies ar emost likely to "die out" over longer timescales. In order to correlate the datasets in the simulations, each run is started at 9:00 CET. Unfortunately, Planetlab version 2.0 was not mature enough at the time of the experiments to be able to run experiments on 130 different processors. However, the job times of runs that are performed on the same node mostly show different characteristics (see also [11]). For these reasons, we use those different datasets as if they were performed on two different homogeneous nodes

at the same site.

We divided the datasets of the 130 runs into two sets of datasets: one set contains 40 datasets and the other 90 datasets. In order to compare the results of [8] with the simulation results of this paper, the first set consists of datasets which are generated from the same nodes as in that paper. The second set contains datasets generated from in total 22 different nodes which includes datasets generated from the same nodes which are used for set one. We note that set two does not contain copies of datasets of set one. Set one only consists of nodes in the USA: Boston, Pasadena, Salt Lake City, San Diego, Tucson, and Washington DC. The second set contains, besides the datasets which are generated on the same nodes as set one, datasets which are generated on the following additional nodes: Amsterdam, The Netherlands; Cambridge, UK; Beijing, China; Copenhagen, Denmark; Le Chesnay, France; Madrid, Spain; Moscow, Russia; Santa Barbara, USA; Seoul, South Korea; Singapore; Sydney, Australia; Tel Aviv, Israel; Taipei, Taiwan (Academica Sinica); Taipei, Taiwan (National Taiwan University); Vancouver, Canada; and Warsaw, Poland.

The job times in set one are on average approximately 72500 ms, and in set two 65000 ms. Further analysis shows that the job times on the nodes in set two show more burstiness and have higher differences between the average job times on the processors. That last property is mainly caused by the fact that the nodes in set two are globally distributed and the nodes in set one are distributed within the USA; set one shows more coherence between the generated datasets. In order to work with the different job time measurements, we define $JT_i(k)$ as the $k$th job time measurement of dataset $i$, where $i = 1 \ldots 130$ and $k = 1 \ldots 2000$. The first 40 datasets form set one and the last 90 form set two.

To provide a strong basis of the simulations, extensive measurements on the durations of the other different processes besides the job times (as explained in Section 2) of a BSP program are necessary. The application for the measurements has been carefully chosen so as to meet the following requirements. The application must have the same dependencies between its iterations as a general BSP program and the same structure between the processors. A suitable application is the Successive Over Relaxation (SOR) application. SOR is an iterative method that has proven to be useful in solving Laplace equations. For more information, we refer to [12].

For our simulations, we need realistic measurements of the synchronization times ($ST$s) for the simulations of ELB, DLB, and JR. We discovered that the synchronization time strongly depends on the maximum of the send times between all pairs of neighbor processors. Therefore, for realistic simulations we need send times measurements between all possible pairs of nodes. Analysis of the send times have shown that the send times between two nodes do not depend on the number of processors used. Consequently, in total 77 original

SOR-application runs on four processors were necessary to generate datasets of the send times between each possible pair of nodes. In total 231 datasets of 2000 send times have been created during this process. The send times are on average around 750 ms. We define $SndT_{i,j}(k)$ as the $k$th send time measurement between node $i$ and $j$, where $i, j = 1 \ldots 130$, and $k = 1 \ldots 2000$.

Further, it is important for the JR simulation to gather realistic measurements of the time that the finalize message takes to be sent from the fastest node to another node. Therefore, we implemented this send process in the SOR-application. This process is different from the above send process, because less information has to be sent, and no acknowledgement is needed. We ran again in total 77 original SOR-application runs on four processors to generate 231 datasets of 2000 finalize-message times. The finalize message times were on average around 300 ms. We define $FM_{i,j}(k)$ as the $k$th measurement of the finalize-message send-time between node $i$ and $j$, where $i, j = 1 \ldots 130$, and $k = 1 \ldots 2000$.

Moreover, for the DLB simulation it is essential to gather measurements of the rescheduling time. Consequently, we implemented the complete rescheduling phase in the DLB application. Analyses have shown that it is sufficient to randomly select in the simulation rescheduling times ($RSchT$s) from a set of 10000 measurements. The $RSchT$ depends on too many different factors to subdivide the $RSchT$s to all those factors. The total rescheduling procedure, as explained in 2.2.1, takes on average around 37500 ms. We note that it is possible to apply more effective packaging methods in this procedure, which can significantly decrease the $RSchT$s. Define $RSchT(l)$ as the $l$th measurement of the rescheduling time, where $l = 1 \ldots 10000$.

In addition, we use the data of the DLB rescheduling times to estimate the overhead to switch from one to the other implementation type. Those times will in practice show comparable charateristics for the following reason. During an implementation switch, the steps to be taken are the same as during as DLB rescheduling phase (details in: 2.2.3) A switch from JR to DLB takes less time than a DLB rescheduling phase because the amount of the to be redistributed load is smaller; the nodes already contain most of the necessary load of the other processors. Small experiments have shown that a switch from JR to DLB takes around 60% of the time of a DLB rescheduling step. During a switch from DLB to JR, more data has to be redistributed due to the fact that all the processors need to gather replications of load from the other processors. This switch takes around 140% of the time of a DLB rescheduling step.

Altogether, we generated the following datasets for our trace-driven simulation analyses: $JT_i(k)$, with $i = 1 \ldots 130$, $k = 1 \ldots 2000$, $SndT_{i,j}(k)$, $FM_{i,j}(k)$, and $RSchT(l)$, with $i = 1 \ldots 22$, $j = 1 \ldots 22$, $k = 1 \ldots 2000$, and $l = 1 \ldots 10000$.

### 3.2 Simulation details

Before we explain the simulation details of DLB, and JR, we define for both strategies:

$$\begin{aligned} D(P) &:= \text{runtime of DLB-run on } P \text{ nodes} \\ R(R,P) &:= \text{runtime of } R\text{-JR run on } P \text{ nodes} \\ R^*(P) &:= \text{runtime best JR strategy on } P \text{ nodes} \\ &:= \min_{R=1,2,4,\ldots,P} R(R,P). \end{aligned}$$

Furthermore, we define the speedups of DLB and of the best JR strategy as the number of times that those strategies are faster than the run without DLB or JR: ELB.

$$\text{speedup } D(P) := \frac{R(1,P)}{D(P)}, \tag{1}$$

$$\text{speedup } R^*(P) := \frac{R(1,P)}{R^*(P)}. \tag{2}$$

Note that $R(1,P)$, which is the running time of a 1-JR run (i.e., each job exists only one time), equals the running time of a ELB run.

**Simulations of dynamic load balancing**

In this section, we describe the details of the trace-driven DLB simulations. We assume a linear relation between the job size and their job times in BSP programs. The following steps have been incorporated in the simulations:

**Step 1:** We randomly select a resource set $S = \{p_1, \ldots, p_P\}$ of $P$ processors from the datasets and order them such that the send times between the processors are minimized. These numbers $p_1, \ldots, p_P$ correspond to the numbers of the datasets.

**Step 2:** The DES-based prediction $\hat{y}_i$ (see for more details: [10]) represents the predicted job time on processor $i$. Consequently, the expected speed of processor $i$ (i.e., the fraction of the total load processed per ms) for the next iteration is $\frac{1}{P\hat{y}_i}$. Further, the expectation of the total processor speed of the $P$ processors together is $\sum_{i=1}^{P} \frac{1}{P\hat{y}_i}$. Consequently, the expected time of the next iteration without send- and rescheduling times given an optimal load distribution is:

$$\frac{1}{\frac{1}{P}\sum_{i=1}^{P} \frac{1}{\hat{y}_i}}. \tag{3}$$

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

$$\frac{P}{\hat{y}_i \sum_{i=1}^{P} \frac{1}{\hat{y}_i}} \tag{4}$$

Finally, to simulate the $EJT_i$ of processor $i$ for this and the next iterations before the next load rescheduling step, we multiply the fractions with the real corresponding data values $JT_i$ from the datasets. We introduce parameter $k$ which indicates the iteration number.

$$EJT_i(k) = \frac{P}{\hat{y}_i(k) \sum_{i=1}^{P} \frac{1}{\hat{y}_i(k)}} JT_i(k). \tag{5}$$

**Step 3:** Next, we derive the iteration time (IT) which is the maximum of the $EJT_i$s of that iteration plus the synchronization time (ST):

$$IT(k) = \max_{i=1,\ldots,P} EJT_i(k) + ST(k). \tag{6}$$

**Step 4:** We derive the running time of the $R$-JR run by repeating step three 2000 times, sum up all the ITs, and add up all the load rescheduling times:

$$D(P) := \sum_{k=1}^{2000} IT(k) + \sum_{l=1}^{\lfloor 2000/N \rfloor} RSchT(l), \tag{7}$$

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

**Step 5:** We derive the expected running time of a DLB run on $P$ processors by repeating steps one to four 1000 times, and finally computing the average of the running times.

**Simulations of job replication**

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

**Step 1:** See step one of the DLB simulation.

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

**Step 3:** In this step, we derive the effective job times $(EJT_1, \ldots, EJT_P)$ for all $P$ processors. Therefore, we first derive within each execution group which processor finished the same job as first. This can be done

by taking one job time value from each dataset of that execution group and observe which processor has the lowest job time. Within one exection group the $EJT$ of the fastest processor ($f$), $EJT_f$, equals the job time value from its dataset ($JT_f$). The $EJT$s of the other processors in the same execution group equal $EJT_f$ plus the time that it takes to send the finalize-message from $f$ to the other processors in the execution group, which is the above defined $FM_{f,j}$.

$$EJT_i = \begin{cases} \min_{j \in EG(i)} JT_j & \text{when } i = f, \\ \min_{j \in EG(i)} JT_j + FM_{f,i} & \text{else.} \end{cases} \tag{8}$$

**Step 4:** Next, we derive the iteration time ($IT$). This time can be derived by repeating step two $R$ times (each processor gets $R$ different jobs during one iteration), sum up the $EJT$s for each processor, taking the maximum of those sums, and adding up the synchronization time ($ST$). Note that it is necessary to take into account all the previous $EJT$s of each processor, because of the dependencies between consecutive $EJT$s. We introduce parameters $k$, and $m$, which respectively indicate the iteration number and the step number within a iteration. Given the above definitions, the iteration time equals:

$$IT(k) = \max_{i=1,\ldots,P} \sum_{m=1}^{R} EJT_i(k,m) + ST(k). \tag{9}$$

**Step 5:** We derive the running time of the $R$-JR run by repeating step three until all datavalues of the $R$ datasets have been processed in the simulation and sum up all the $IT$s. In order to be able to compare the running times of runs with different values for $R$, we multiply this sum with $R$ to derive a comparable running time of 2000 iterations for each possible $R$-JR run:

$$R(R,P) := R \times \sum_{k=1}^{\lfloor 2000/R \rfloor} IT(k). \tag{10}$$

**Step 6:** We derive the expected running time of a $R$-JR run on $P$ processors by repeating steps one to five 1000 times, and finally computing the average of the running times.

### 3.3 Dynamic selection method

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

### 3.3.1 Analysis

To be able to develop such a selection method, we need to find formulas that indicate the height of the iteration times for the different implementations for a given easy-to-measure statistic. To this end, we first derive an approximation of the expected IT for DLB, which is based on the predictions $\hat{y}_t$, by adding the expected ST and rescheduling time to (3).

$$\mathbb{E}IT_{D(P)} \approx \frac{1}{\frac{1}{P}\sum_{i=1}^{P}\frac{1}{\mathbb{E}\hat{y}_i}} + \mathbb{E}ST + \frac{\mathbb{E}RSchT}{N}. \tag{11}$$

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

$$\mathbb{E}IT_{D(P)} = a_{DLB}\frac{1}{\frac{1}{P}\sum_{i=1}^{P}\frac{1}{\mathbb{E}\hat{y}_i}} + b_{DLB}. \tag{12}$$

Furthermore, because of the reason that the iteration time of JR also has a strong linear relation to (3), we define the expected iteration time of JR as:

$$\mathbb{E}IT_{R(R,P)} = a_{R(R,P)}\frac{1}{\frac{1}{P}\sum_{i=1}^{P}\frac{1}{\mathbb{E}\hat{y}_i}} + b_{R(R,P)}. \tag{13}$$

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

Next, we investigate the relation between the average of 100 realized iteration times provided by realistic trace-driven simulations and the estimations of the ITs by the above equations. An effective statistical property to quantify this dependency is the correlation coefficient. This property can be derived by substituting the data values of both quantities in the correlation formula. The correlation coefficient varies by definition between -1.0, which indicates a complete negative linear dependency, and 1.0, which indicates a complete positive linear dependency. A correlation of 0.0 indicates no linear dependency. More details can be found in [15]. The high correlation of 0.97 between those values implies that equations (12) to (13) are accurate indications of the possibly realizable speedups. Consequently, this means that for a given implementation and a given choice of parameters (e.g., number of processors), the speedup strongly depends on the statistic, which is defined in (3). In the rest of this paper, we call this statistic $Y$ ( $= P/\sum_{i=1}^{P}\frac{1}{\mathbb{E}JT_i}$).

We consider the following situation. We perform 1000 simulations of a 4-JR implementation on 4 nodes and 1000 simulations of a DLB implementation on 4 nodes. For comparison reasons, the 1000 simulations of the DLB implementation have been executed on the same set of nodes as on the corresponding JR simulation. Figure 2 depicts the averages of 100 iteration times of those trace-driven simulated runs of DLB and JR against statistic $Y$.

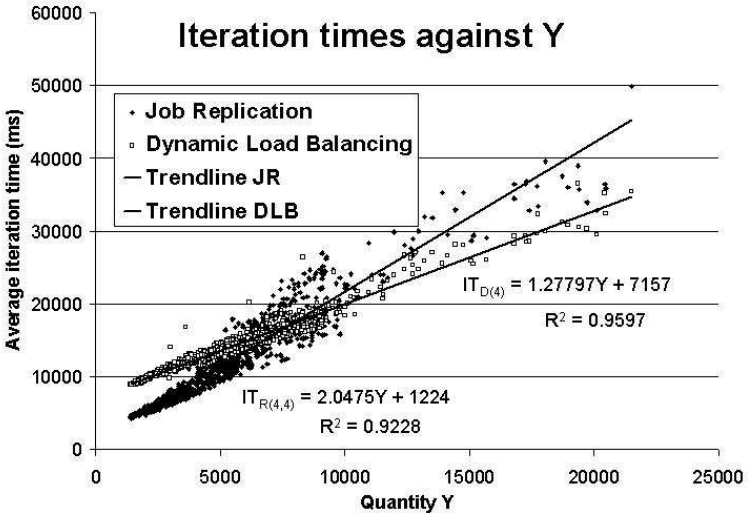



Figure 2: Scatterplot of DLB and JR iteration times

This figure shows, as expected, the strong linear relation between the statistic $Y$ and the ITs. We fitted trendlines by a least-square fit and derived the $R^2$ values of those equations, which will both be explained at the end of this section. The trendline of the ITs of JR has the following equation:

$$IT_{R(4,4)} = 2.0475Y + 1224, \tag{14}$$

with a $R^2$-value of 0.9228, and for DLB the trendline equals:

$$IT_{DLB(4)} = 1.2779Y + 7157, \tag{15}$$

with a $R^2$-value of 0.9597.

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

$IT_{R(4,4)}$ with $IT_{DLB(4)}$.

**The DLB/JR method**

Given the above equations, we are able to develop a method that dynamically chooses the most effective implementation from both DLB or JR. We propose the following selection method:

**Step 1:** Start with DLB as the current choice.

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

**Step 3:** Estimate the job- and iteration times for the other as the current implementation by straightforward computations, which are shown in 3.2.

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

**Step 5:** Fit the values of $a_{DLB}$, $a_{R(R,P)}$, $b_{DLB}$, and $b_{R(R,P)}$ in equations (12)-(13) by a least-square fit (more information below) of the collected data about the ITs and the statistics [20]. When only one datapoint has been collected, go to step eight and take the ITs of the first $I_p$ iterations as expected ITs for the next $I_p$ iterations.

**Step 6:** To derive an expectation of the ITs of the different implementations, take the latest computed value of $Y$ and substitute it in equations (12)-(13) with the fitted values of $a_{DLB}$, $a_{R(R,P)}$, $b_{DLB}$, and $b_{R(R,P)}$.

**Step 7:** Choose the implementation with the lowest expected IT for the next $I_p$ iterations.

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

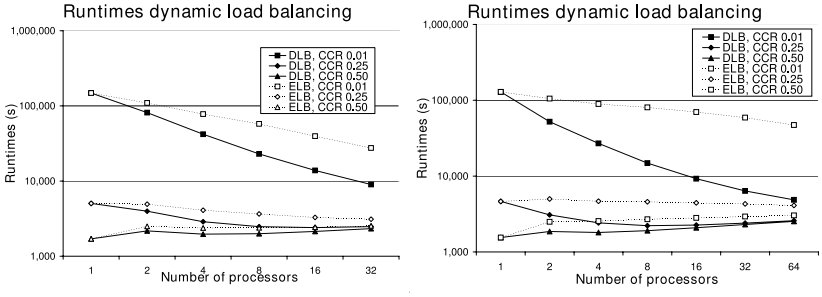
A least-square fit is an effective method that fits a linear equation in a collection of datapoints. It is based on minimizing the sum of the squares of the deviations between the linear equation and the datapoints. The values of $a$ and $b$ can be derived by a direct formula of the values of the data points. The $R^2$ value is an indication of the overall deviation between the trendline and the datapoints, and ranges between 0.0 (no fit) and 1.0 (complete fit). For brevity, we omitted the formulas for $a$, $b$, and the $R^2$. For more details, we refer to [15].

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

The results of extensive analysis of IT-predictions have shown that an estimation which is based on substituting measurements of $Y$ in (12)-(13) is far more accurate than taking the average IT of the last 100

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

Table 1: Sets, $R$s and CCRs, for given $P$




(a) **40 datasets of USA nodes** (b) **90 datasets of global nodes**

Figure 3: DLB Runtimes for different CCRs

iterations.

# 4 Experimental Results

In this section, we present the experimental results of the trace-driven simulation experiments. We consider simulations of the previously described different methods to cope with the fluctuating job times on the processors: (1) equal load balancing (ELB), (2) dynamic load balancing (DLB), (3) job replication (JR), and (4) the selection method. We perform extensive simulations in which we investigate several experimental settings. As described in Section 3, we use the two generated sets of datasets: one set of 40 datasets and one set of 90 datasets.

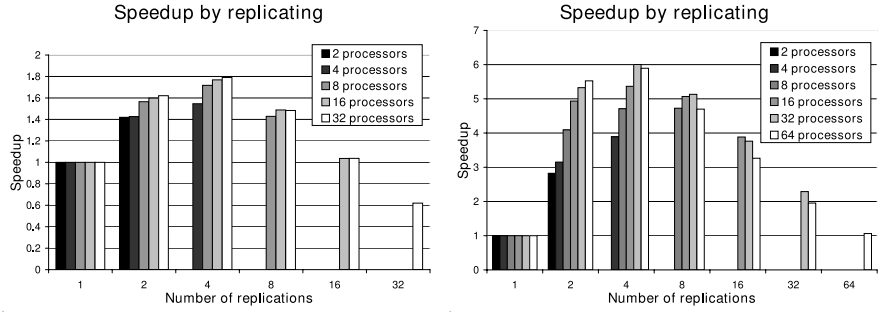
First, we simulate the running times of DLB for different numbers of processors with set one and two. Furthermore, we analyze the impact of the CCR, defined as the communication time divided by the computation time, on the runtimes of DLB. The average CCR of a ELB run on 2 processors is found to be 0.01 in our datasets. This value indicates that the computations take around 99% of the total run time. In order to investigate this impact, we linearly interpolate the heights of the computation times such that we are able to derive simulations of runs with a CCR of 0.25, and of 0.50. With those interpolated job times, we again simulate runs based on DLB for a wide range of situations.

Second, we simulate runs of BSP parallel applications that use JR and analyze the expected speedups for different numbers of processors, for different numbers of replications, for the two different sets of datasets, and for the following different CCR values: 0.01, 0.25, and 0.50. Subsequently, we derive the optimal number of replications for the different situations. Table 1 depicts which different situations have been investigated.

Third, we compare the results of the running times and the speedups of the ELB-, DLB- and the JR imple-

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

Table 2: $R^*$s for given $P$s




(a) **40 datasets of USA nodes** (b) **90 datasets of global nodes**

Figure 4: Speedups of JR

mentations.

Fourth, we simulate the speedups of the proposed selection method which is described in 3.3. Therefore, we perform trace-driven simulations of this method with set two of datasets.

### 4.1 DLB experiments

In this section, we present the results of the simulations of the DLB runs. We investigated the DLB running-times with both sets of processors for runs with a CCR of 0.01, 0.25, and 0.50 on 1, 2, 4, 8, 16, and 32 processors. Figure 3(a) depicts the average runtimes on a logaritmic scale of all performed simulations on nodes of set one. Moreover, Figure 3(b) depicts the average runtimes on a logaritmic scale of all the performed simulations on nodes of set two.

From the simulations results of the runs with a CCR of 0.01, we conclude that selecting more processors in the run decreases the running times, which is the main motivation for programming in parallel. Although the rescheduling- and send times increase when more processors are selected in the run, the decrease in the computation times for this case is always higher. As is shown by Figures 3(a) and 3(b), we draw different conclusions when the CCR is higher. For runs on nodes of set one and a CCR of 0.25, we notice a decrease in running times until the amount of 16 processors is selected. When more processors have been selected, the running times will increase due to the significant heights of the rescheduling- and send times. Furthermore, we conclude that for every experimental setting, DLB consistently shows a speedup in comparison to ELB, even for runs with a CCR of 0.50. However, one could doubt the effectiveness of programming in parallel, because of the increase in running times when more processors are used.

### 4.2 Job Replication experiments

In this section, we show the results of the JR experiments. Figure 4(a) and 4(b) depict the speedups in the running times of JR on different numbers of processors with CCR 0.01 (the original CCR) for set one and two respectively. For example, the white bars show the speedup that can be gained by JR for different $R$s on 32 processors. Subsequently, we present in Table 2 the optimal number of replications for different numbers of processors.

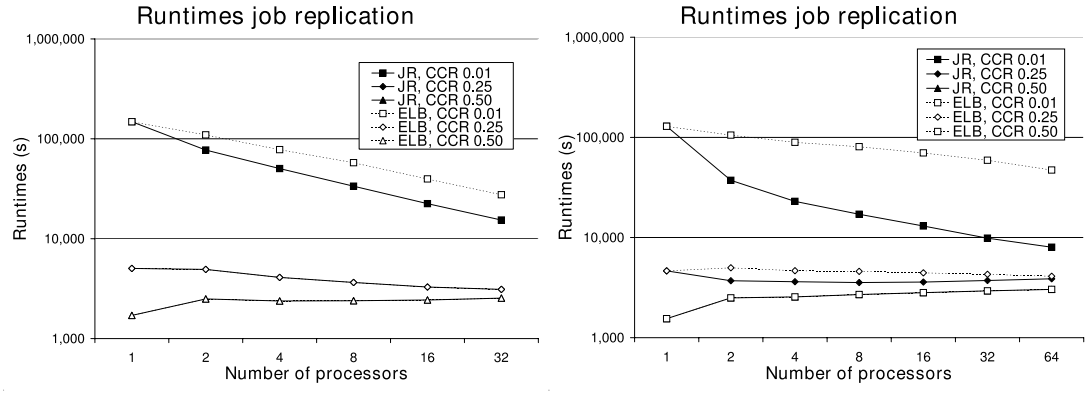
We conclude from Figure 4(a) and Table 2 that for a given number of processors, the speedup increases when 2-JR or 4-JR has been applied. The impact of the fluctuations on the running times is high enough such that 4 replications of each job (i.e., make 3 extra copies of each job) have to be made to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

The results of the set with 90 datasets, which are shown by Figure 4(b) and Table 2, show again 4 as the optimal number of replications for most numbers of processors. Except for the runs with 8 processors, as can be seen in Figure 4(b), a slightly higher speedup can be gained for the 8- in comparison with the 4-JR case. A difference between the results of this set and the results of the first set is that the speedups are significantly higher. For example, the highest speedup for set one is below the 2.0, while for set two even speedups of higher than 6.0 have been registered. This is caused by the differences between set one and two, which is described in 3.1.

Furthermore, we simulated the running times of JR on parallel applications with a CCR of 0.25 and 0.50. Figure 5(a) and 5(b) present the results. We conclude that JR on parallel applications with a CCR of 0.50 never leads to decreases in the running times; the best replication strategy is not to replicate, which equals an ELB run. Furthermore, JR for the nodes in set one and a CCR of 0.25, JR again does not lead to a running-time decrease. However, the running times on nodes of set two and a CCR of 0.25 show in many cases a small decrease of 30% compared to ELB for the best JR strategy. The runtimes of runs with CCR of 0.50 show a consistent increase in running times when the number of processors increases, which shows that running in parallel in this case is not effective.

### 4.3 Comparison of ELB, DLB, and JR

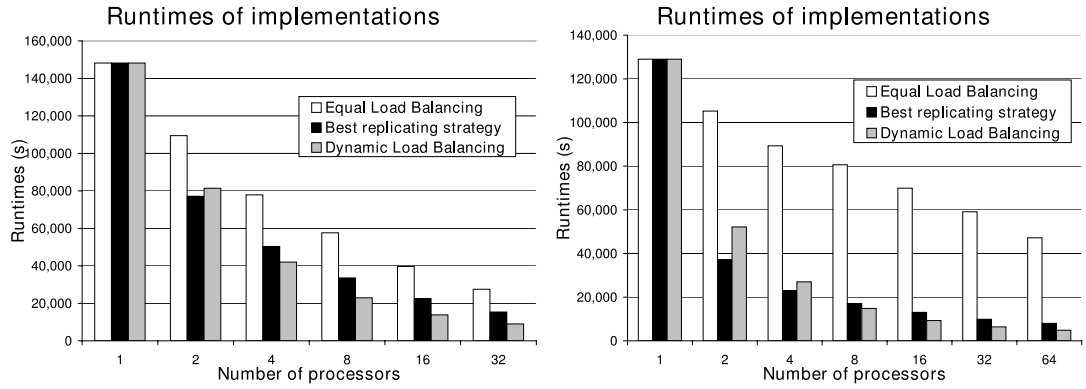
In this section, we compare the running times of the best JR strategy, the DLB implementation, and of the ELB. Figures 6(a) and 6(b) depict the running times of the above mentioned three different strategies for




(a) **40 datasets of USA nodes**

(b) **90 datasets of globally distributed nodes**

Figure 5: Running times of JR for different CCR processors set one and set two respectively.



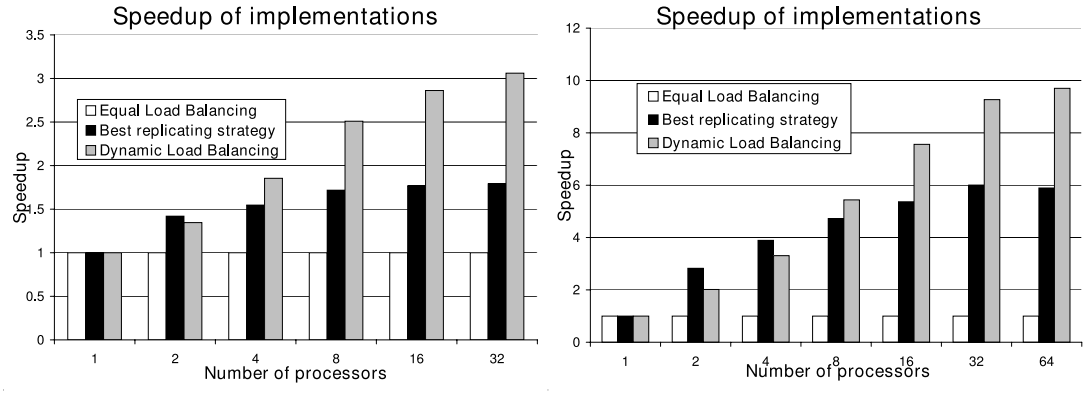

(a) **40 datasets of USA nodes**

(b) **90 datasets of globally distributed nodes**

Figure 6: Runtimes of DLB and JR with CCR 0.01

From Figures 6(a) and 6(b) can be concluded that the running times of JR and of DLB are consistently lower than those of the ELB implementation. For a CCR of 0.01 holds that for all three types of implementations and for both sets of datasets deploying more processors for the same amount of load leads to a significant speedup. Figure 6(b) depicts superlinear speedups for JR and DLB if two processors are used instead of one. This is caused by the effect that if one processor is used, peaks in the job times have a dramatic impact on the total runtime of the application. In runs with two processors this effect can be reduced by the faster second processor. A difference between the results of set one and set two is that the running times of the DLB and JR implementations of set two decrease faster while the running times of ELB decrease slower. This is caused by the differences between set one and two, which is described in 3.1.

For further analysis, we compute the speedups, as defined in 3.2, of the best JR strategy and DLB for set one and two from the running times from Figures 6(a) and 6(b). Figures 7(a) and 7(b) depict those computed speedups.




(a) **40 datasets of USA nodes**

(b) **90 datasets of globally distributed nodes**

Figure 7: Speedups of DLB and JR with sets of 40 and 90 datasets with CCR 0.01

Figure 7(a) shows for the simulations with set one that DLB consistently outperforms or at least performs as good as JR. We conclude from Figure 7(b) that in comparison with the results of nodes from set one significantly higher speedups can be gained on the nodes of set two. This insight corresponds to the observations mentioned above in this section. In 3.1, the causes of these observationed have been mentioned. Moreover, we notice for the experiments with the nodes of set two that the best JR strategy has a higher speedup than DLB for runs on 2 or 4 processors. However, when more processors are used, DLB outperforms all JR strategies. This is mainly caused by the fact that when more processors are used, the amount of load per processor decreases and, as a consequence, the load that has to be redistributed during the DLB rescheduling phase decreases. For this reason, the overhead time of DLB shows more sensitivity to the number of processors. Those numbers are the results of the following trade-off. On the one hand, when more processors have been deployed, the load per processor and, therefore, the gain by DLB, decreases. On the other hand, the probability on slow processors increases, which delay the whole process in ELB implementations. We observe that the results of set one show that the speedup of DLB has its maximum at 16 processors and for set two the maximal speedup can be gained when 32 processors are used. This is again caused by the differences in average processing times between the nodes and the higher fluctuations over time. We remark that the results of Figure 7(a) are consistent with the results in [9]: the DLB runs on four randomly selected processors of set one show again on average a speedup of 1.8.

Comparing the results of DLB and JR for the different CCR 0.01, 0.25, and 0.50 show that for a CCR of

0.01, there are some circumstances for which replication shows the best results. However, for CCRs of 0.25 and 0.50 DLB clearly outperforms JR for all situations. For many of the cases, replication does not even show speedups in comparison with ELB. We conclude that when the CCR increases, the gain that can be obtained by JR is too small the compensate the overhead and extra computations of this method. On the other hand, the overhead of DLB remains low enough when the CCR increases, to be able to gain speedups.

### 4.4 Selection-method experiments

In this section, we present the results of the trace-driven simulations of the selection-method implementation. This method selects dynamically during the run between the two implementations DLB and JR. The details of this method are described in 3.3. For example, we perform an experiment with DLB, and 4-JR on 4 processors. Figure 8 shows us for this experiment the derived values of statistic $Y$ for the following different groups of iteration numbers: $1-100, 101-200, \ldots, 1901-2000$. Moreover, the Figure depicts the corresponding realized iteration times of DLB and JR.

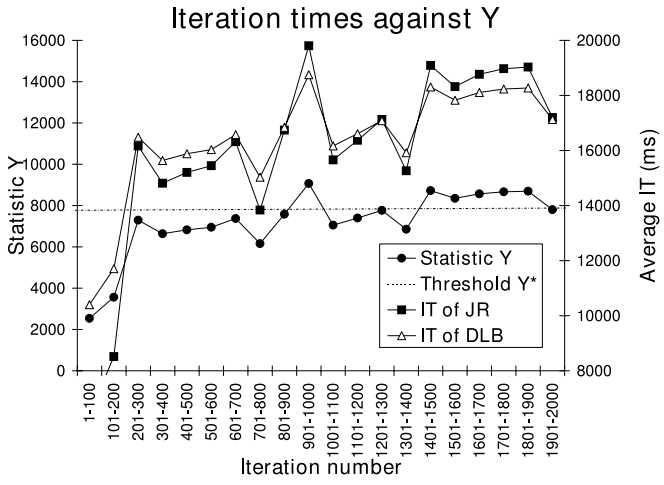



Figure 8: Statistic $Y$ against ITs of DLB and JR

As we have seen above, the threshold value $Y^*$ for the comparison between a JR-run with 4 replications and a DLB run on 4 processors is 7708. Figure 8 shows that for this situation, the $Y$ is lower of equal than 7708 until iteration number 900. This means that JR has the lowest running times, which corresponds to the measured average ITs for these iteration numbers. Until iteration number 1400, the statictic $Y$ moves around the threshold value and therefore both implementations can be used. Likewise, the realized ITs of both implementations do not differ significantly. After iteration number 1400, the threshold value clearly moves above the threshold which indicates that DLB is the best choice, because of lower iteration times.

Finally, we compare the speedups of the selection method to those of the DLB and JR implementations. To this end, we performed 1000 experiments with the selection method on 1, 2, 4, 8, 16, 32, and 64 randomly

chosen nodes from set two. Figure 9 depicts the speedups of the method compared to those of DLB and JR.

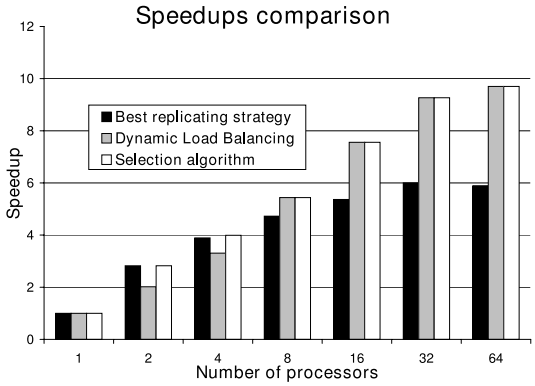



Figure 9: Speedup of selection method, DLB and JR

We conclude that the method that selects between DLB and JR performs at least as good as both DLB and JR for all situations. The overhead of the switches to the best performing method is in every experimental setting completely compensated by the gain in running time resulting from those effective switches. For the cases in which one of the two implementation types is significantly faster, the performance of the selection method exactly equals the highest possible performance, because for those situations it immediately selects the one with the highest speedups. Altogether, the results in Figure 9 show that the introduced dynamic method is very effective in making Bulk Synchronous Processing parallel programs robust against the fluctuations of a globally distributed grid environment and in which there is no knowledge about which of the methods JR or DLB will perform best.

# 5 Conclusion and Outlook

In summary, in this paper we have made an extensive assessment and comparison between the two main techniques that are most suitable to make parallel applications robust against the unpredictability of the grid: DLB and JR. We found that there exists an easy-to-measure statistic $Y$ and a corresponding threshold value $Y^*$ such that DLB outperforms JR for $Y > Y^*$, whereas JR consistently performs better for $Y < Y^*$. Based on this observation, we propose the so-called DLB/JR method, a simple and easy-to-implement approach that can make on-the-fly decisions about whether to use to DLB or JR. Extensive simulations based on a large set of real data in a global-scale grid show that this new dynamic approach always performs at least as good as both DLB and JR in all circumstances. As such, the DLB/JR approach presented provides a powerful means to make parallel applications robust in large-scale grid environments.

The results presented in this paper address a number of challenges for further research. First, the ex-

perimental results presented are based on trace-driven simulations. The next step is to bring the results to a higher level of reality by extensively analyzing the effectiveness of our approach for a variety of "live" global-scale grid environments. Second, a challenging area for further research is to make use of mathematical techniques to provide a more solid foundation for the results presented in this paper, e.g., by formally proving the increased effectiveness induced by the our approach. Finally, an interesting and important problem is to determine the optimal number of job replicas needed to obtain the best speedup performance. Our experimental results suggest that generating four job replicas seems to be a good value to start with. It remains a challenging topic for further research to develop practical guidelines for determining the (near-)optimal replication level in large-scale grid environments.

## References


[1] http://www.planet-lab.org.

[2] I. Ahmad and Y.-K. Kwok (1994). A New Approach to Scheduling Parallel Programs Using Task Duplication. In: *Proc. International Conference on Parallel Processing*, pp. 47-51.

[3] H. Attiya (2004). Two Phase algorithm for load balancing in heterogeneous distributed systems. In: *Proc. 12th Euromicro conference on parallel, distributed and network-based processing*, pp. 434.

[4] R. Bajaj and D.P. Agrawal (2004). Improving Scheduling of Tasks in a Heterogeneous Environment. *IEEE Transactions on Parallel and Distributed Systems* **15** (2), pp. 107-118.

[5] I. Banicescu and V. Velusamy (2002). Load Balancing Highly Irregular Computations with the Adaptive Factoring", In: *Proc. of the 16th International Parallel and Distributed Processing Symposium* (IEEE Computer Society), pp.195.

[6] J.Y. Colin and P. Chretienne (1991). C.P.M. Scheduling with Small Communication Delays and Task Duplication. *Operations Research* **39** (4), pp. 680-486.

[7] S. Darbha and D.P. Agrawal (1998). Optimal Scheduling Algorithm for Distributed-Memory Machines. *IEEE Transactions on Parallel and Distributed Systems* **9** (1), pp. 87-95.

[8] A.M. Dobber, G.M. Koole and R.D. van der Mei (2004). Dynamic Load Balancing for a Grid Application. In: *Proc. HiPC 2004* (Springer-Verlag), 342-352.

[9] A.M. Dobber, G.M. Koole and R.D. van der Mei (2005). Dynamic Load Balancing Experiments in a Grid. In: *Proc. 5th IEEE International Symposium on Cluster Computing and the Grid* (IEEE Press), 123–130.

[10] A.M. Dobber, R.D. van der Mei and G.M. Koole (2006). Effective Prediction of Job Processing Times in a Large-Scale Grid Environment. In: *Proc. 15th IEEE International Symposium on High Performance Distributed Computing*.

[11] A.M. Dobber, R.D. van der Mei and G.M. Koole (2006). Statistical Properties of Task Running Times in a Global-Scale Grid Environment. In: *Proc. 6th IEEE International Symposium on Cluster Computing and the Grid* (IEEE Press).

[12] D.J. Evans (1984). Parallel SOR iterative methods. *Parallel Computing* **1**, 3-18.

[13] M.J. Flynn (1972). Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers* **C-21**, 948-960.

[14] L. Guodong, C. Daoxu, D. Wang and Z. Defu (2003). Task Clustering and Scheduling to Multiprocessors with Duplication. In: *Proc. International Parallel and Distributed Processing Symposium*, pp. 6b.

[15] J.F. Kenney and E.S. Keeping (1962). Mathematics of Statistics, chapter 15: Linear and Correlation, pp. 252-285.

[16] A. Mondal, K. Goda and M. Kitsuregawa (2003). Effective load-balancing via migration and replication in spatial grids. LNCS 2736", pp. 201-211.

[17] G.-L. Park, B. Shirazi and J. Marquis (1998). Mapping of parallel tasks to multipro0cessors with duplication. In: *Proc. 31st Annual Hawaii Intenational Conference on Systems Sciences*, vol. 7, pp. 96.

[18] B.A. Shirazi, A.R. Hurson and K.M. Kavi (1995). *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE CS Press.

[19] L.G. Valiant (1990). A bridging model for parallel computation. *Communications of the ACM* **33** (8), pp. 103-111.

[20] D. York (1966). Least-square fitting of a straight line. *Canadian Journal of Physics* **44**, pp. 1079-1086.

[21] K. Yu-Kwong (2000). Parallel program execution on a heterogeneous PC cluster using task duplication. In: *Proc. 9th Heterogeneous Computing Workshop*, pp. 364.

[22] M.J. Zaki, W. Li and S. Parthasarathy (1997). Customized dynamic load balancing for a network of workstations. *Journal of Parallel and Distributed Comouting* **43** (2), 156-162.

[23] S. Zhou (1988). A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering* **14** (9), 1327-1341.