Int J Parallel Prog
DOI 10.1007/s10766-014-0309-6

A Framework to Analyze the Performance of Load Balancing Schemes for Ensembles of Stochastic Simulations

Tae-Hyuk Ahn · Adrian Sandu · Layne T. Watson · Clifford A. Shaffer · Yang Cao · William T. Baumann

Received: 5 June 2013 / Accepted: 6 March 2014

Abstract Ensembles of simulations are employed to estimate the statistics of possible future states of a system, and are widely used in important applications such as climate change and biological modeling. Ensembles of runs can naturally be executed in parallel. However, when the CPU times of individual simulations vary considerably, a simple strategy of assigning an equal number of tasks per processor can lead to serious work imbalances and low parallel efficiency. This paper presents a new probabilistic framework to analyze the performance of dynamic load balancing algorithms

T.-H. Ahn (✉)
Computer Science and Mathematics Division, Oak Ridge National Laboratory,
Oak Ridge, TN 37831, USA
e-mail: ahnt@ornl.gov

A. Sandu · C. A. Shaffer · Y. Cao
Department of Computer Science, Virginia Polytechnic Institute and State University,
Blacksburg, VA 24061, USA
e-mail: sandu@cs.vt.edu

C. A. Shaffer
e-mail: shaffer@cs.vt.edu

Y. Cao
e-mail: ycao@cs.vt.edu

L. T. Watson
Departments of Computer Science, Mathematics, and Aerospace and Ocean Engineering,
Virginia Polytechnic Institute and State University,
Blacksburg, VA 24061, USA
e-mail: ltw@cs.vt.edu

W. T. Baumann
Department of Electrical and Computer Engineering, Virginia Polytechnic Institute
and State University, Blacksburg, VA 24061, USA
e-mail: baumann@vt.edu

Published online: 01 April 2014

for ensembles of simulations where many tasks are mapped onto each processor, and where the individual compute times vary considerably among tasks. Four load balancing strategies are discussed: most-dividing, all-redistribution, random-polling, and neighbor-redistribution. Simulation results with a stochastic budding yeast cell cycle model are consistent with the theoretical analysis. It is especially significant that there is a provable global decrease in load imbalance for the local rebalancing algorithms due to scalability concerns for the global rebalancing algorithms. The overall simulation time is reduced by up to 25 %, and the total processor idle time by 85 %.

Keywords Dynamic load balancing (DLB) · Probabilistic framework analysis · Ensemble simulations · Stochastic simulation algorithm (SSA) · High-performance computing (HPC) · Budding yeast cell cycle

1 Introduction

Important scientific applications like climate and biological system modeling incorporate stochastic effects in order to capture the variability of the real world. For example, biological systems are frequently modeled as networks of interacting chemical reactions. At the molecular level, these reactions evolve stochastically and the stochastic effects typically become important when there are a small number of molecules for one or more species involved in a reaction [25]. Systems in which the stochastic effects are important must be described statistically.

The easiest way to generate statistics for complex systems is to run ensembles of simulations using different initial conditions and parameter values; their results sample the probability density of all possible future states [26]. Taking advantage of the ideally parallel nature of ensembles, individual runs can be easily distributed to different processors. However, the inherent variability in compute times among individual simulations can lead to considerable load imbalances. For these simulations, load balancing among processors is necessary to avoid wasting computing resources and power.

A large body of research literature is available on static and dynamic load balancing (DLB) techniques [5,9,19,20,23]. Two classes of DLB methods are widely used: scheduling (work-sharing) schemes [13,18,27] and work-stealing schemes [6,21,32]. The factoring approach, one of the classical scheduling algorithms, allocates large chunks of iterations at the beginning of the computation to reduce scheduling overhead, and dynamically assigns small chunks towards the end of the computation to achieve good load balancing [13]. The work-stealing approach identifies and moves tasks from overloaded processors to idle processors. A simple yet powerful work-stealing scheme is random polling [15]. A processor that runs out of assigned work sends requests to randomly chosen processors, until a busy one is found. The requestee then sends part of its work to the requestor. Scheduling schemes usually take a centralized load balancing approach where the remaining tasks are stored in a central work queue [15,37]. Work-stealing schemes, on the other hand, can employ both centralized and decentralized load balancing approaches [15]. In centralized DLB a master process distributes tasks to the workers (slave processes). In decentralized DLB, tasks are moved between peer processes.

This paper focuses on several work-stealing DLB methods and their application to stochastic biochemical simulations. In the most-dividing (MD) algorithm, the processor that finishes first receives new tasks from the most overloaded processor. In the all-redistribution (AR) algorithm, when one worker becomes idle, all remaining jobs are evenly redistributed among all processors. In the random-polling (RP) algorithm new tasks are received from a randomly chosen processor. In the neighbor-redistribution (NR) algorithm, the idle processor and its neighbors redistribute evenly all remaining jobs (on the neighbor processors). The Dijkstra-Scholten algorithm [12] and the Shavit-Francez algorithm [34] are adapted for detecting termination. MD and AR use a centralized DLB approach, whereas RP and NR employ a decentralized one.

A number of papers available in the literature have considered probabilistic analyses of task scheduling. In their classical work Kruskal and Weiss [22] provide the first probabilistic analysis of allocating independent subtasks on parallel processors. They consider the case where running times of the subtasks are independent identically distributed (i.i.d.) random variables with mean and standard deviation (μ, σ). Batches of tasks of fixed size are allocated dynamically from a central work pool. Lucco [24] considers the problem of scheduling chunks of tasks from a central work pool. He proposes the Probabilistic Tapering method which computes the sizes of chunks of tasks that keep the load imbalance bounded with high probability. Scheduling overheads are incorporated into the theoretical analysis. Hagerup [16] considers a centralized work pool and introduces the heuristic strategy to allocate batches to processors in such a way as to achieve a small expected overall finishing time. Bast [3,4] has the goal to allocate jobs from a central work pool such as to minimize the total wasted time (the sum of all delays plus the idle times of processors). Her approach is not probabilistic, but deterministic, and is based on estimated lower and upper bounds for processing times of chunks of a given size.

All these previous studies [3,4,16,22,24] perform a probabilistic analysis for *scheduling algorithms*, where chunks of tasks are assigned to processors from a central work pool. Scheduling a chunk of jobs to a node changes the load of that node, but leaves all the other loads unchanged. The current work also assumes task run times that are i.i.d. random variables like in [22,24]. However, *the current work extends the probabilistic analysis to dynamic load balancing algorithms*. The theoretical problem is more complex than that considered in [3,4,16,22,24] since each load balancing step changes the work load of both the work donor(s) and the work receiver(s). We consider several dynamic load balancing strategies (most-dividing, all-redistribution, random-polling, and neighbor-redistribution), out of which the last two use a fully distributed work pool. There is renewed interest in scheduling and dynamic load balancing algorithms in the era of new technologies including cloud computing system [29,30,39]. Recent work focuses on developing and testing new balancing schemes, but there are few new results available to date on probabilistic theoretical analysis.

The novelty of the work presented in this paper consists of a new *general framework* for analyzing *work-stealing dynamic load balancing algorithms* when applied to large ensembles of stochastic simulations. In this case the established deterministic analysis approaches are not appropriate, so a probabilistic analysis is developed. The times per task are assumed to be independent identically distributed random variables with a certain probability distribution. This is a natural assumption for ensemble

computations, where the same model is run repeatedly with different initial conditions and parameter values. No assumption is made, however, about the shape of the underlying probability density function; the proposed analysis is very general. The level of load imbalance (defined by a given metric) is also a random variable. The analysis focuses on quantifying the decrease in the *expected value* of the random load imbalance. The probabilistic analysis reveals that the four applied DLB methods are effective for moderate parallelism; scalability is not investigated here. While the performance analysis is complex, the four DLB methods described here are easy to implement. Numerical results show that they achieve considerable savings in computation time for a computational biology application. The relative performance of the four DLB strategies is analyzed numerically for a biological problem in Sect. 5.

The paper is organized as follows. The four load balancing algorithms are presented in Sect. 2. Section 3 explains the analysis framework, and Sect. 4 contains the probabilistic analysis of the load balancing algorithms. Section 5 shows theoretical and experimental results with a cell cycle model. Section 6 draws some conclusions.

2 Load Balancing Algorithms

This section introduces four different dynamic load balancing strategies.

2.1 Motivation

Each run of a stochastic simulation leads to different results. The goal of running an ensemble of stochastic simulations is to estimate the probability distribution of all possible outcomes. This typically requires thousands of simulations run concurrently on many CPUs. The stochastic nature of the system and the potentially dramatic differences running time per simulation can cause a severe load imbalance among processors that are running many simulations.

Consider, for example, stochastic simulations of the budding yeast cell. For certain mutants, a cell might never divide, or it might always divide, with some probability. Therefore, the CPU time to run the simulation is quite different from one case to another. Figure 1. shows 100 prototype mutant multistage cell lineage simulations assigned statically to 10 worker processors. The results reveal a considerable load imbalance, with the CPUs being idle for approximately 40 % of the aggregate compute time. This results in poor utilization of computer resources, longer time to results, and reduced scientific productivity. Dynamic load balancing strategies are required to improve the parallel efficiency. The stochastic simulation algorithm and budding yeast cell cycle model are explained in detail in Sect. 5.

2.2 Most-Dividing (MD) Algorithm

The most-dividing (MD) algorithm is based on the *central redistribution* work of Powley [28] and Hillis [17]. The idea of the MD algorithm is presented in Fig. 2a. First, the tasks (cell simulations) are evenly distributed to every worker processor in

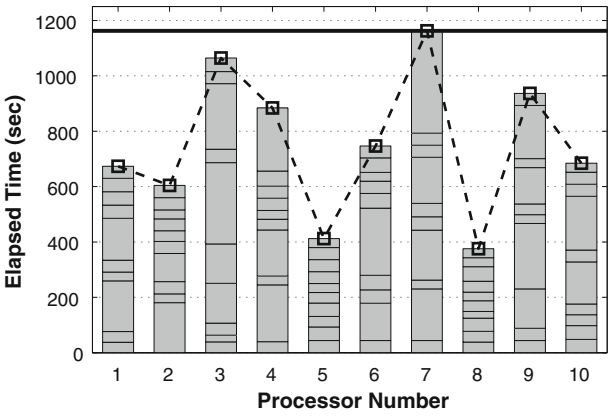

Fig. 1 Elapsed compute times for 100 prototype mutant multistage cell cycle simulations by static distribution across ten worker processors. *Dotted line* represents different CPU times per processor and the *solid line* indicates the wall clock time

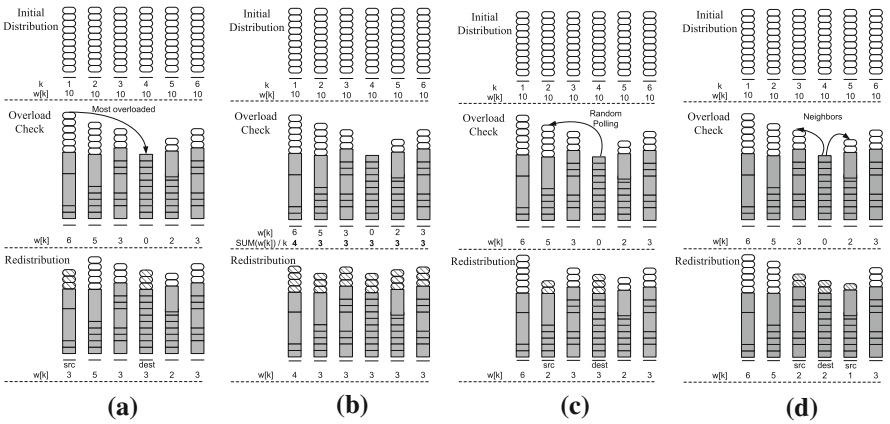

Fig. 2 Adaptive load balancing strategies. *Ellipses* represent tasks to be done and *gray rectangles* represent completed tasks. *Right diagonal patterned ellipses* indicate tasks to be done on processors whose load has been adjusted by an adaptive load balancing algorithm. **a** MD DLB idea, **b** AR DLB idea, **c** RP DLB idea, **d** NR DLB idea

the system. Workers concurrently execute their jobs. Due to different CPU times per task, other processors may be well behind the first processor to finish its tasks. The processor that finishes its jobs becomes idle. The processor with the largest number of remaining jobs is considered to be the most overloaded processor. At this time the most overloaded processor sends out half of its remaining jobs to the idle processor. This sequence of steps is executed repeatedly until there is no remaining work.

To implement the MD algorithm, the idle processor has to receive new work from the highest load processor. Therefore, the highest load processor stops its work, and reduces its remaining work when another processor has completed all of its work. Stopping the computation when all the tasks are completed is called *termina-*

tion. The Dijkstra-Scholten algorithm [12] and the Shavit-Francez algorithm [34] are adapted for detecting terminations using requests and acknowledgement messages. Initially, each processor is in one of two states: inactive and active. Upon receiving a task from the master, worker processors are active. Workers send a message to the master whenever they finish a job, and receive messages setting their state to continue activity or become inactive once the termination condition is satisfied. When any processor finishes its assigned jobs, the highest load processor receives a suspend message. It suspends execution after finishing the currently active job, reduces its tasks to half of its remaining jobs, and then resumes execution where it left off.

2.3 All-Redistribution (AR) Algorithm

The all-redistribution (AR) method is also a centralized load balancing scheme. The idea of the AR algorithm is presented in Fig. 2b. The initial step of the AR algorithm is similar to that of the MD algorithm. The processor that finishes its jobs first becomes idle, and notifies the master of its idle status. Then, the master directs all workers to suspend execution, redistributes all remaining jobs in the workers' queues evenly among all workers, and finally directs the workers to resume execution.

2.4 Random-Polling (RP) Algorithm

Centralized schemes are inherently limited in terms of scalability. Due to finite communications resources, bottlenecks appear when many worker processors request jobs simultaneously from the same master. One approach to solve the scalability issue is to organize the system into multiple master/worker partitions, which are supervised by a dedicated supermaster process. Another approach, the decentralized scheme, is to fully distribute and execute tasks on all processors without any master supervision.

The random-polling (RP) method is a receiver-initiated decentralized load balancing algorithm [37]. Figure 2c illustrates the idea. When a worker processor becomes idle, it randomly polls other processors until it finds a busy one. The busy worker becomes a donor and sends out half of its remaining jobs to the idle processor. Each processor is selected as a donor with equal probability, ensuring that work requests are evenly distributed.

The implementation of the RP algorithm associates with each processor one of the following three states: available, idle, and locked. A processor with remaining jobs beyond the active one is in the available state. A processor that finishes its jobs becomes idle. A processor with one (active) job is locked. An idle processor randomly polls other processors to request jobs. Upon receiving the request, an available processor agrees to become a donor. The state of the donor processor(s) changes from available to locked in order to avoid overlaps (i.e., to become a donor for multiple idle processors that happened to randomly poll it). After the RP load balancing step ends, a locked processor is released and becomes available if there are remaining jobs besides the currently active one.

2.5 Neighbor-Redistribution (NR) Algorithm

The idea of the neighbor-distribution (NR) decentralized load balancing scheme is presented in Fig. 2d. A processor that finishes its jobs informs its neighbors of its idle status. The set of neighbors is predefined based on the network topology of the system (in this paper a 2-D torus topology is considered for the numerical experiments). From the algorithmic perspective, the sets of neighbors can be arbitrarily defined; assume that each processor has $k - 1$ neighbors. The idle processor and its neighbors redistribute evenly all their remaining jobs (i.e., apply the AR algorithm on the subset of k processors).

The NR load balancing step performs a local redistribution of jobs, and therefore is suitable for parallel architectures where groups of nodes are linked directly. In this case the NR method is related to the dimension exchange algorithm, where a dimension corresponds to a fully connected subset [5,10,38]. Similar to RP, the NR algorithm uses three states (available, idle, and locked) to avoid overlaps (i.e., participation by the same processor in the balancing steps performed by two distinct groups of neighbors).

3 The Analysis Framework

This section presents a probabilistic framework for load balancing analysis. The assumptions needed for the analysis and the metrics used to measure load imbalance are considered.

3.1 Assumptions for the Analysis

The computational goal is to run an ensemble of n stochastic (biochemical) simulations. Each individual simulation is referred to as a "task". Due to the stochastic nature of each simulation, the execution time t associated with a particular task cannot be estimated in advance. (The same situation occurs with deterministic adaptive models where the grid or time step adaptation depends on the data, and the chosen grid and step sizes greatly affect the total compute time.) The task compute times are modeled by random variables.

Assumption 1 The compute times associated with different tasks are independent identically distributed (i.i.d.) random variables.

The mean and the standard deviation of the random variable task compute time T are denoted by μ_T and σ_T, respectively. The exact shape of the probability density function for T is not relevant for the analysis; thus, the analysis results are very general.

Assumption 1 naturally covers the case where the ensemble is obtained by running the same model multiple times, with different initial conditions, different parameter values, or different seeds of the pseudo random number generator. New model runs are independent of the results of previous runs. Assumption 1 is also appropriate where multiple models are being run, and where each model of the batch is chosen with a specified frequency.

Next, the mapping of the n tasks of the ensemble onto the p processors is considered. Processor i has R_i tasks, such that $R_1 + \cdots + R_p = n$. Let t_{ij} denote the compute time of the jth task on the ith processor where $i = 1, \ldots, p, \quad j = 1, \ldots, R_i$. Note that all t_{ij} are i.i.d. random variables according to Assumption 1. The total compute time $X_i = \sum_{j=1}^{R_i} t_{ij}$ of processor i is also a random variable. In probability theory, the central limit theorem (CLT) states that the normalized sum of a sufficiently large number of independent identically distributed random variables, each with finite mean and variance, will be approximately standard normally distributed [31]. Therefore, using Assumption 1, if R_i is large enough, then X_i will be approximately normally distributed with

$$\mathrm{E}[X_i] = R_i \cdot \mu_T, \quad \mathrm{Var}[X_i] = R_i \cdot \sigma_T^2.$$

It is therefore assumed that

Assumption 2 The number of tasks mapped onto each processor is sufficiently large such that the probability density function of the total compute time per processor is approximately Gaussian.

Assumption 2 allows the analysis to work with Gaussian distributions of the total compute times per processor regardless of the underlying distribution of individual task times. Thus a very general setting for the analysis is possible. Assumption 2 is invalid during the winddown period (when there are only a few tasks left per processor), but that is a small fraction of the total ensemble computation time. Even during winddown load balancing continues to be beneficial, but the theoretical analysis cannot be directly applied.

3.2 Metrics of Load Imbalance

The algebraic mean of the compute times per processor is defined as $\eta_X = \frac{1}{p}\sum_{i=1}^{p} X_i = \frac{1}{p}\sum_{i=1}^{p}\sum_{j=1}^{R_i} t_{ij}$. Note that η_X is itself a random variable with $\mathrm{E}[\eta_X] = (n/p)\,\mu_T$. The algebraic variance of the compute times among processors is defined by $\xi_X^2 = \frac{1}{p-1}\sum_{i=1}^{p}(X_i - \eta_X)^2$

and is also a random variable. The square root of the algebraic variance (RAV), $\sqrt{\xi_X^2}$, is also considered. The basic premise of variance is that larger variance between the compute times on different processors is a symptom of larger load imbalance. The first measure of the degree of load imbalance is therefore the expected value of the algebraic variance,

$$\mathrm{E}\left[\xi_X^2\right] = \frac{1}{p-1}\sum_{i=1}^{p}\mathrm{E}\left[(X_i - \eta_X)^2\right], \tag{1}$$

or more conveniently the square root $\sqrt{\mathrm{E}\left[\xi_X^2\right]}$.

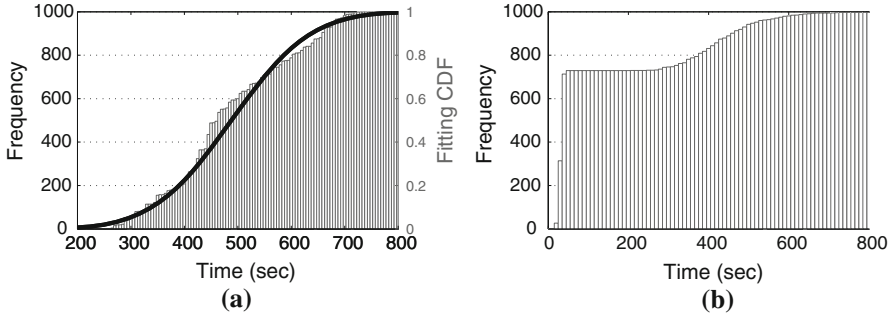

Fig. 3 Discrete cumulative histogram of compute times per cell (*bar*) for wild-type and mutant simulations. The *solid line* represents the best-fit Gaussian CDF. **a** Wild-type 1,000 simulations, **b** Prototype mutant 1,000 simulations

Consider now the minimum and the maximum computation times among all processors, $Y_1 = \min\{X_1, \ldots, X_p\}$ and $Y_p = \max\{X_1, \ldots, X_p\}$. These are both random variables. The idle time spent by processor i is the difference between the maximum time and the compute time on the processor, $Y_p - X_i$. The second measure of load imbalance is the expected value of the largest idle time, i.e., the difference between the largest and the smallest compute times across all processors,

$$\mathrm{E}\left[Y_p - Y_1\right] = \mathrm{E}\left[\max\{X_1, \ldots, X_p\}\right] - \mathrm{E}\left[\min\{X_1, \ldots, X_p\}\right]. \tag{2}$$

Finally, the third measure of load imbalance is the expected value of the average idle compute time across all processors,

$$\mathrm{E}\left[\frac{1}{p}\sum_{i=1}^{p}\left(Y_p - X_i\right)\right] = \mathrm{E}\left[Y_p - \eta_X\right]. \tag{3}$$

3.3 Variability in Compute Times Per Cell

The wild-type cell lineage simulation time distribution from a simulation experiment is plotted in Fig. 3a. This distribution is based on 1,000 budding yeast multistage cell tracking simulations. The best continuous Gaussian CDF approximation to the discrete cumulative histogram is also shown; it is clear that the cell cycle simulation times are not normally distributed [35]. The wild-type simulation data from Fig. 3a has the mean and standard deviation

$$\mu_T = 488.1 \text{ sec.} \quad \text{and} \quad \sigma_T = 116.6 \text{ sec.} \tag{4a}$$

Figure 3b shows the cumulative discrete histogram of 1,000 prototype budding yeast mutant simulations. Approximately 75 % of the cells never divide and the remaining 25 % divide very irregularly. For the mutant simulation results

$$\mu_T = 152.0 \text{ sec.} \quad \text{and} \quad \sigma_T = 191.1 \text{ sec.} \tag{4b}$$

4 Analysis of the Dynamic Load Balancing Algorithms

Level of load imbalance is measured by three well-defined metrics (1)–(3). The analysis approach quantifies the *expected value* of the load imbalance metrics before and after each work redistribution step, and assess the reduction in the expected load imbalance.

4.1 Order Statistics

Let $X_1, \ldots, X_p$ be p independent identically distributed random variables with a probability density function (PDF) $f_X(x)$, and cumulative distribution function (CDF) $F_X(x)$. The variables $Y_1 \leq Y_2 \leq \cdots \leq Y_p$, where the Y_i are the X_i arranged in order of increasing magnitudes, are called *order statistics* corresponding to the random sample $X_1, \ldots, X_p$. Therefore, $Y_1 = \min\{X_1, \ldots, X_p\}$ and $Y_p = \max\{X_1, \ldots, X_p\}$. Some useful facts about order statistics [11] follow. The CDF of the largest order statistic Y_p is given by

$$F_{Y_p}(y) = \Pr[\, Y_p \leq y \,] = \prod_{j=1}^{p} \Pr[\, X_j \leq y \,] = [\, F_X(y) \,]^p$$

because the X_js are independent. Likewise

$$F_{Y_1}(y) = \Pr[\, Y_1 \leq y \,] = 1 - [\, 1 - F_X(y) \,]^p.$$

Thus, the special probability density function for the maximum Y_p and the minimum Y_1 are

$$f_{Y_p}(y) = p\,[F_X(y)]^{p-1}\, f_X(y), \tag{5a}$$

$$f_{Y_1}(y) = p\,[1 - F_X(y)]^{p-1}\, f_X(y). \tag{5b}$$

Numerical evaluation of expected order statistics is complex. Chen and Tyler [7] show that the expected value, standard deviation, and complete PDF of the extreme order distributions can be accurately approximated when the samples X_i are i.i.d. Gaussian. The formulas use the expression $\Phi^{-1}\left(0.5264^{1/p}\right)$, where p is the sample size and $\Phi^{-1}(y) = \sqrt{2}\,\mathrm{erfinv}(2y-1)$ is the inverse function of the standard Gaussian CDF Φ, and erfinv is the inverse of the error function $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$. Specifically, the expected values of the largest and the smallest order statistics of i.i.d. Gaussian samples are, respectively,

$$\mathrm{E}[Y_p] \approx \mu_X + \sigma_X\, \Phi^{-1}\left(0.5264^{1/p}\right), \tag{6a}$$

$$\mathrm{E}[Y_1] \approx \mu_X - \sigma_X\, \Phi^{-1}\left(0.5264^{1/p}\right). \tag{6b}$$

Numerical evidence presented in [7] indicates that the relative approximation errors are of the order of a few percent for moderately large values of p ($p \geq 20$). Note that the compute times X_i here are not identically distributed (unless all the R_i are the same), and thus in general (6) does not apply to the min and max compute times Y_1 and Y_p. (6) is used only for initially equal R_i followed by AR, and in that case experimental results presented in Sect. 5 indicate that the approximations (6a) and (6b) are very close to the experimentally determined expected values.

4.2 Some Useful Results for Load Balancing

Consider the moment right after one processor (say, P_1) finishes all its jobs. Define R_i to be the number of remaining jobs outstanding (including the one currently executing) on the processor P_i. Since the analysis is carried out at a given moment in time, the R_i are known and are not random variables. Let t_{ij} be the execution time for the remaining job j on processor P_i. Let X_i be the execution time of all the remaining jobs on P_i.

Consider a load balancing step that redistributes (nonexecuting) jobs among processors. Since the total number of jobs is not changed, the algebraic mean of compute times remains the same.

Lemma 1 *Let $X = [X_1, \ldots, X_p]$ be the remaining compute times when the first processor finishes its tasks, and before the load balancing is performed. Let $X' = [X'_1, \ldots, X'_p]$ be the vector of compute times after the load balancing step. The algebraic mean of compute times per processor is the same random variable for all configurations,*

$$\eta_X = \frac{1}{p}\sum_{i=1}^{p} X_i = \eta_{X'} = \frac{1}{p}\sum_{i=1}^{p} X'_i,$$

since X' contains the same tasks, therefore the same execution times t_{ij}, as X (just distributed differently). Therefore a load balancing step does not change the expected algebraic mean time $\mathrm{E}[\eta_X] = \mathrm{E}[\eta_{X'}]$.

In what follows, the algebraic mean and the algebraic variance of the remaining number of jobs per processor are denoted by

$$\mathrm{M}(R) = \frac{1}{p}\sum_{\ell=1}^{p} R_\ell,\ \mathrm{V}(R) = \frac{1}{p-1}\sum_{i=1}^{p}\big(R_i - \mathrm{M}(R)\big)^2. \tag{7}$$

Lemma 2 estimates the time left to completion.

Lemma 2 *Consider a task that has started but not yet finished. There is no information about how far along the computation is. The total execution time t of the task is a random variable from a distribution with mean μ_T and variation σ_T^2. Then the total remaining execution time τ is a random variable with*

$$\mathrm{E}[\tau] = \frac{1}{2}\mu_T, \quad \mathrm{Var}[\tau] = \frac{\sigma_T^2}{3} + \frac{\mu_T^2}{12}.$$

Proof Consider that a fraction $f \in [0, 1]$ of the task still needs to run, while a fraction $(1 - f)$ of the task has completed. Since there is no information about the part that is done, f is a uniformly distributed random variable, $f \in \mathcal{U}([0, 1])$. It is important to notice that t and f are independent random variables.

The time left to completion $\tau = f\,t$ is a random variable. Due to the independence of t and f,

$$\mathrm{E}[\tau] = \mathrm{E}[f\,t] = \mathrm{E}[f]\,\mathrm{E}[t] = \frac{1}{2}\mu_T.$$

For the variance,

$$\mathrm{E}\left[\left(f\,t - \frac{1}{2}\,\mu_T\right)^2\right] = \frac{\sigma_T^2}{3} + \frac{\mu_T^2}{12}.$$

□

Define adjusted numbers $\widehat{R}_i$ of tasks per processor such that $\mathrm{E}[X_i] = \widehat{R}_i\,\mu_T$. The definition must account for the fact that one task may be running. When all processors are still working, one task on each processor is running. The adjusted number of tasks is defined as

$$\widehat{R}_i = R_i - \frac{1}{2} \quad \text{for } i = 1, \ldots, p. \tag{8a}$$

Assume, without loss of generality, that P_1 is the first processor that finishes its jobs and becomes idle. All other processors have one running task, and therefore

$$\widehat{R}_1 = 0 \quad \text{and} \quad \widehat{R}_i = R_i - \frac{1}{2} \quad \text{for } i = 2, \ldots, p. \tag{8b}$$

Right after the load balancing step the processor P_i has R'_i tasks to execute. On processors $P_2, \ldots, P_p$ the first task is the one being executed, but all the R'_1 tasks on P_1 are newly assigned and queued: none has started yet. This leads to

$$\widehat{R}_1 = R'_1 \quad \text{and} \quad \widehat{R}_i = R'_i - \frac{1}{2} \quad \text{for } i = 2, \ldots, p. \tag{8c}$$

The following lemma is a useful ingredient in proving the main results of the paper.

Lemma 3 *The expected value of the algebraic variance of the compute times* (1) *depends on both the algebraic variance of the number of tasks, and the variance of the individual compute times, and is given by*

$$\mathrm{E}\left[\xi_X^2\right] = \mathrm{V}(\widehat{R})\,\mu_T^2 + \mathrm{M}(\widehat{R})\,\sigma_T^2 + \frac{p-1}{p}\left(-\frac{1}{6}\,\sigma_T^2 + \frac{1}{12}\,\mu_T^2\right), \tag{9}$$

where the $\widehat{R}_i$ *represent the adjusted numbers of tasks per processor* (8)*. The algebraic mean* $\mathrm{M}(\widehat{R})$ *and the algebraic variance* $\mathrm{V}(\widehat{R})$ *are defined in* (7)*.*

Proof Redefine t_{ij} to be the time remaining for job j on processor P_i; the (random) compute times per processor and their average are

$$X_i = \sum_{j=1}^{R_i} t_{ij}, \quad \eta_X = \frac{1}{p} \sum_{\ell=1}^{p} \sum_{m=1}^{R_\ell} t_{\ell m}.$$

Each processor P_i, $i \geq 2$, has one task in progress with expected completion time $\mu_T/2$ when P_1 finishes its tasks. Note that if P_1 is idle (right before load balancing) then $R_1 = 0$. If P_1 is not idle (right after load balancing step) then none of the tasks assigned to it has started and $\mathrm{E}[t_{1j}] = \mu_T$ for $j = 1, \ldots, R_1$. Consequently, the mean compute time of the first job is different on P_1 than it is on other processors;

$$\mathrm{E}[t_{ij}] = \begin{cases} \mu_T/2, & i = 2, \ldots, p \text{ and } j = 1, \\ \mu_T, & i = 2, \ldots, p \text{ and } 2 \leq j \leq R_i, \\ \mu_T, & i = 1 \text{ and } R_1 \geq 1, \\ 0\,, & i = 1 \text{ and } R_1 = 0. \end{cases}$$

Now

$$\begin{aligned} X_i - \eta_X &= \sum_{j=1}^{R_i} t_{ij} - \frac{1}{p} \sum_{\ell=1}^{p} \sum_{m=1}^{R_\ell} t_{\ell m} = \left(1 - \frac{1}{p}\right) \sum_{j=1}^{R_i} t_{ij} - \frac{1}{p} \sum_{\ell=1, \ell \neq i}^{p} \sum_{m=1}^{R_\ell} t_{\ell m} \\ &= \left(1 - \frac{1}{p}\right) \sum_{j=1}^{R_i} \left(t_{ij} - \mathrm{E}[t_{ij}]\right) + \left(1 - \frac{1}{p}\right) \sum_{j=1}^{R_i} \mathrm{E}[t_{ij}] \\ &\quad - \frac{1}{p} \sum_{\substack{\ell=1 \\ \ell \neq i}}^{p} \sum_{m=1}^{R_\ell} (t_{\ell m} - \mathrm{E}[t_{\ell m}]) - \frac{1}{p} \sum_{\substack{\ell=1 \\ \ell \neq i}}^{p} \sum_{m=1}^{R_\ell} \mathrm{E}[t_{\ell m}]. \end{aligned} \tag{10}$$

Recall $\widehat{R}_i$ was defined so that $\sum_{j=1}^{R_i} \mathrm{E}[t_{ij}] = \widehat{R}_i\, \mu_T$, and

$$\left(1 - \frac{1}{p}\right) \sum_{j=1}^{R_i} \mathrm{E}[t_{ij}] - \frac{1}{p} \sum_{\substack{\ell=1 \\ \ell \neq i}}^{p} \sum_{m=1}^{R_\ell} \mathrm{E}[t_{\ell m}] = \left(\widehat{R}_i - \mathrm{M}(\widehat{R})\right) \mu_T.$$

Note that $\mathrm{E}\left[\sum_{j=1}^{R_i} \left(t_{ij} - \mathrm{E}[t_{ij}]\right)\right] = 0$. $\mathrm{E}[(X_i - \eta_X)^2]$ will be determined from (10). First apply Lemma 2 to get

$$\mathrm{E}\left[\left(t_{ij} - \mathrm{E}[t_{ij}]\right)^2\right] = \begin{cases} \frac{\sigma_T^2}{3} + \frac{\mu_T^2}{12}, & i = 2, \ldots, p \text{ and} j = 1, \\ \sigma_T^2, & i = 2, \ldots, p \text{ and} j \geq 2, \\ \sigma_T^2, & i = 1 \text{and } R_1 \geq 1, \\ 0\,, & i = 1 \text{ and } R_1 = 0. \end{cases}$$

In compact notation

$$\sum_{j=1}^{R_i} \mathrm{E}\left[\left(t_{ij} - \mathrm{E}[t_{ij}]\right)^2\right] = \widehat{R}_i\,\sigma_T^2 + \left(-\frac{1}{6}\,\sigma_T^2 + \frac{1}{12}\,\mu_T^2\right)(1-\delta_{i1}),$$

where δ_{i1} is the Kronecker delta. Due to the independence of individual compute times, $\mathrm{E}\left[\left(t_{ij} - \mathrm{E}[t_{ij}]\right)\left(t_{\ell m} - \mathrm{E}[t_{\ell m}]\right)\right] = 0$ for $j \neq m$ or$i \neq \ell$. Hence

$$\begin{aligned}\mathrm{E}[(X_i - \eta_X)^2] = {} & \left(\widehat{R}_i - \mathrm{M}(\widehat{R})\right)^2 \mu_T^2 + \frac{1}{p}\,\left(\mathrm{M}(\widehat{R}) + (p-2)\,\widehat{R}_i\right)\sigma_T^2 \\ & + \left(-\frac{1}{6}\,\sigma_T^2 + \frac{1}{12}\,\mu_T^2\right)\left(\frac{p^2 - p - 1 - (p^2 - 2p)\,\delta_{i1}}{p^2}\right).\end{aligned}$$

Finally the expected value of the algebraic variance

$$\begin{aligned}\mathrm{E}\left[\xi_X^2\right] &= \frac{1}{p-1}\sum_{i=1}^{p}\mathrm{E}[(X_i - \eta_X)^2] \\ &= \mathrm{V}(\widehat{R})\,\mu_T^2 + \mathrm{M}(\widehat{R})\,\sigma_T^2 + \frac{p-1}{p}\,\left(-\frac{1}{6}\,\sigma_T^2 + \frac{1}{12}\,\mu_T^2\right).\end{aligned}$$

□

Lemma 3 provides insight into how the load balancing algorithms reduce the algebraic variance of compute times per processor. Any redistribution of tasks does not change the total number of tasks, and therefore does not change the algebraic mean $\mathrm{M}(\widehat{R})$. The second and the third terms in (9) are invariant with any load balancing algorithm. However, a reduction in the algebraic variance $\mathrm{V}(\widehat{R})$ of the number of tasks will decrease the expected algebraic variance of the compute times by reducing the first term in (9). Therefore the following corollary can be derived.

Lemma 4 *Let R and R' be the number of tasks per processor before and after a load redistribution step, respectively. Let X and X' be the compute times per processor before and after a load redistribution step, respectively. The decrease in the expected value of the algebraic variance of the compute times (1) is*

$$\mathrm{E}\left[\xi_X^2\right] - \mathrm{E}\left[\xi_{X'}^2\right] = \left(\mathrm{V}(\widehat{R}) - \mathrm{V}(\widehat{R'})\right)\mu_T^2, \tag{11}$$

where the $\widehat{R}_i$ represent the adjusted numbers of tasks per processor (8).

4.3 Analysis of Static Distribution

Let X_i be total job execution time for processor i and t_{ij} be the jth job time of X_i in the static (no dynamic load balancing) approach. Assume the total number n

of jobs is a multiple of the number p of processors. Processor i is assigned $R = \lceil n/p \rceil = n/p$ jobs, so that $X_i = \sum_{j=1}^{R} t_{ij}$ for $i = 1, \ldots, p$. From the analysis in the previous section, the total times per processor are i.i.d. approximately Gaussian random variables $X_1, \ldots, X_p$ with mean and variance given by

$$\mu_X = R\,\mu_T, \quad \sigma_X^2 = R\,\sigma_T^2. \tag{12}$$

The expected value of the algebraic variance (1) is given by Eq. (9) where all $\widehat{R}_i = R$,

$$\mathrm{E}\left[\xi_X^2\right] = R\sigma_T^2 + \frac{p-1}{p}\left(-\frac{1}{6}\sigma_T^2 + \frac{1}{12}\mu_T^2\right). \tag{13}$$

Let Y be the order distribution of $X : Y_1 \le Y_2 \le \cdots \le Y_p$. From (5a)–(5b),

$$\mathrm{E}[Y_p] = \int_{-\infty}^{\infty} y\; p\,[F_X(y)]^{p-1} f_X(y)\; dy, \tag{14a}$$

$$\mathrm{E}[Y_1] = \int_{-\infty}^{\infty} y\; p\,[1 - F_X(y)]^{p-1} f_X(y)\; dy \tag{14b}$$

with the Gaussian probability density function $f_X(y)$ and the Gaussian cumulative distribution function $F_X(y)$. From (14a)–(14b) together with the simulation data (4a), the probabilistic load imbalance measures (2)–(3) can be evaluated by numerical integration.

Alternatively, the approximations (6) can be used together with (12) to obtain

$$\mathrm{E}[Y_p - Y_1] \approx 2\sqrt{R}\;\sigma_T\;\Phi^{-1}\left(0.5264^{1/p}\right), \mathrm{E}[Y_p - \eta_Y] \approx \sqrt{R}\;\sigma_T\;\Phi^{-1}\left(0.5264^{1/p}\right).$$

4.4 Analysis of MD Dynamic Load Balancing

Call P_1 the first processor that finishes its jobs and becomes idle. At this time each processor P_i, $i > 1$, has R_i outstanding jobs and a total remaining execution time X_i. By the CLT, each of $X_2, \ldots, X_p$ is approximately normally distributed if all R_i are large. The first (running) job on $P_2, \ldots, P_p$ has a different PDF and a negligible effect on compute time statistics, assuming that $R_i \gg 1$ for $i \ge 2$.

In the MD algorithm the highest loaded processor sends half of its unfinished jobs to the idle processor. Assume, without loss of generality, that and P_p has the highest load of R_p unfinished jobs. The MD load balancing step moves $\lfloor R_p/2 \rfloor$ jobs from the processor P_p to P_1. The loads for $P_2, \ldots, P_{p-1}$ are not changed. Therefore, the number of jobs per processor after redistribution is $R_1' = \left\lfloor \frac{R_p}{2} \right\rfloor$, $R_2' = R_2, \ldots, R_{p-1}' = R_{p-1}$, $R_p' = \left\lceil \frac{R_p}{2} \right\rceil$.

Let X'_i be the remaining compute time for processor P_i after the MD load balancing step. From the above $X'_i = X_i$ for $i = 2, \ldots, p-1$. For the first and last processors the expected values of the compute times are

$$\mathrm{E}[X'_1] = R'_1\,\mu_T = \left\lfloor \frac{R_p}{2} \right\rfloor \mu_T, \quad \mathrm{E}[X'_p] = \left(R'_p - \frac{1}{2}\right)\mu_T = \left(\left\lceil \frac{R_p}{2} \right\rceil - \frac{1}{2}\right)\mu_T.$$

The above expression accounts for the fact that P_p has one task in progress. Furthermore,

$$\mathrm{E}[X'_p] - \mathrm{E}[X'_1] = \left(R_p \bmod 2 - \frac{1}{2}\right)\mu_T.$$

The following propositions prove that each MD redistribution step decreases the level of load imbalance as measured by the metrics (1)–(3).

Proposition 1 *The expected value of the algebraic variance of the compute times per processor* (1) *decreases after a MD DLB step by* $\mathrm{E}[\xi^2_X] - \mathrm{E}[\xi^2_{X'}] = \frac{R_p\,(R_p-1)}{2\,(p-1)}\,\mu_T^2$.

Proof The average adjusted number of tasks per processor is the same before and after MD load balancing, $\mathrm{M}(\widehat{R}) = \mathrm{M}(\widehat{R}')$. The decrease in the algebraic variance of the adjusted number of tasks is

$$\begin{aligned}\mathrm{V}(\widehat{R}) - \mathrm{V}(\widehat{R}') = \frac{1}{p-1}\Big((\widehat{R}_1 - \mathrm{M}(\widehat{R}))^2 - (\widehat{R}'_1 - \mathrm{M}(\widehat{R}))^2 \\ + (\widehat{R}_p - \mathrm{M}(\widehat{R}))^2 - (\widehat{R}'_p - \mathrm{M}(\widehat{R}))^2\Big) = \frac{R_p\,(R_p-1)}{2\,(p-1)}.\end{aligned}$$

Lemma 4 provides the difference between the expected variances of compute times across processors before and after a MD load balancing step,

$$\mathrm{E}[\xi^2_X] - \mathrm{E}[\xi^2_{X'}] = \frac{R_p\,(R_p-1)}{2\,(p-1)}\,\mu_T^2\,. \tag{15}$$

The MD algorithm can be meaningfully applied only when the number of tasks on the most overloaded processor is $R_p \geq 2$. The relation (15) then provides a strict decrease in the expected value of the algebraic variance of compute times. □

Proposition 2 *The expected value of the largest idle time* (2) *is monotonically decreased after a MD DLB step, that is,* $\mathrm{E}\left[Y'_p - Y'_1\right] \leq \mathrm{E}\left[Y_p - Y_1\right]$.

Proof Before the MD load balancing step, the expected maximum imbalance is $\mathrm{E}[Y_p] - \mathrm{E}[Y_1] = \mathrm{E}[Y_p] \geq \mathrm{E}[X_p] = \widehat{R}_p\,\mu_T$.

After the MD load balancing step, the new expected maximum imbalance time is $\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]$. Consider the random variables

$$Z_{min} = \min\{X_2, \ldots, X_{p-1}\}, \quad Z_{max} = \max\{X_2, \ldots, X_{p-1}\} \;\leq\; Y_p.$$

The smallest and the largest order statistics after MD balancing are $Y'_1 = \min\{X'_1, X'_p, Z_{min}\}$ and $Y'_p = \max\{X'_1, X'_p, Z_{max}\}$. There are nine possible combinations of Y'_1 and Y'_p values. Two of them lead to $Y'_1 = Y'_p$, i.e., the maximum idle time is zero after the MD load balancing step. The remaining seven combinations are as follows:

(1) $Y'_1 = Z_{min}$ and $Y'_p = Z_{max}$; (2) $Y'_1 = Z_{min}$ and $Y'_p = X'_p$;
(3) $Y'_1 = Z_{min}$ and $Y'_p = X'_1$; (4) $Y'_1 = X'_p$ and $Y'_p = Z_{max}$;
(5) $Y'_1 = X'_p$ and $Y'_p = X'_1$; (6) $Y'_1 = X'_1$ and $Y'_p = Z_{max}$;
(7) $Y'_1 = X'_1$ and $Y'_p = X'_p$.

In Case (1) the balanced times fall between Z_{min} and Z_{max}. The expected maximum idle time reduction is

$$\begin{aligned}\{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} &= \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Z_{max}] - \mathrm{E}[Z_{min}]\} \\ &= \{\mathrm{E}[Y_p] - \mathrm{E}[Z_{max}]\} + \{\mathrm{E}[Z_{min}]\} \ \geq \mathrm{E}[Z_{min}] \ \geq 0.\end{aligned}$$

The reductions of expected maximum idle times for Cases (2) to (7) are straightforward verification as the Case (1).

$$\begin{aligned}
\text{Case (2):}\ & \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} = \mathrm{E}[Y_p] - \{\mathrm{E}[X'_p] - \mathrm{E}[Z_{min}]\} \\
& \geq \mathrm{E}[Y_p] - \mathrm{E}[X'_p] \geq \widehat{R}_p\, \mu_T - \left(\lceil R_p/2 \rceil - 0.5\right) \mu_T > 0. \\
\text{Case (3):}\ & \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} = \mathrm{E}[Y_p] - \{\mathrm{E}[X'_1] - \mathrm{E}[Z_{min}]\} \\
& \geq \mathrm{E}[Y_p] - \mathrm{E}[X'_1] \geq \left(R_p - 0.5 - \lfloor R_p/2 \rfloor\right) \mu_T \ > \ 0. \\
\text{Case (4):}\ & \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} = \mathrm{E}[Y_p] - \{\mathrm{E}[Z_{max}] - \mathrm{E}[X'_p]\} \\
& \geq \mathrm{E}[X'_p] = \left(\lceil R_p/2 \rceil - 0.5\right) \mu_T \ > \ 0. \\
\text{Case (5):}\ & \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} = \mathrm{E}[Y_p] - \{\mathrm{E}[X'_1] - \mathrm{E}[X'_p]\} \\
& \geq \left(\widehat{R}_p - \lfloor R_p/2 \rfloor + \lceil R_p/2 \rceil - 0.5\right) \mu_T \ > \ 0. \\
\text{Case (6):}\ & \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} = \mathrm{E}[Y_p] - \{\mathrm{E}[Z_{max}] - \mathrm{E}[X'_1]\} \\
& \geq \mathrm{E}[X'_1] \ > \ 0. \\
\text{Case (7):}\ & \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} = \mathrm{E}[Y_p] - \{\mathrm{E}[X'_p] - \mathrm{E}[X'_1]\} \\
& \geq \left(\widehat{R}_p + 0.5 + \lfloor R_p/2 \rfloor - \lceil R_p/2 \rceil\right) \mu_T \ > \ 0.
\end{aligned}$$

Therefore, after a MD load balancing step, the expected maximum time imbalance is always the same or reduced. If $R_2 \geq 1$ and $R_{p-1} \geq 1$, then $\mathrm{E}[Z_{min}] > 0$. Then expected maximum time is always decreased after a MD DLB step. □

Proposition 3 *The expected value of the average idle time* (3) *does not increase after a MD DLB step, that is,* $\mathrm{E}\left[Y'_p - \eta_{X'}\right] \leq \mathrm{E}\left[Y_p - \eta_X\right]$.

Proof The decrease in the expected average idle time is (since $\eta_{X'} = \eta_X$)

$$\mathrm{E}\left[Y_p - \eta_X\right] - \mathrm{E}\left[Y'_p - \eta_{X'}\right] = \mathrm{E}\left[Y_p\right] - \mathrm{E}\left[Y'_p\right].$$

Consider each of the possible values of Y'_p separately.

$$(1) Y'_p = Z_{max}: \ \mathrm{E}[Y_p - Y'_p] = \mathrm{E}\left[Y_p - Z_{max}\right] \geq 0;$$
$$(2) Y'_p = X'_1: \quad \mathrm{E}[Y_p - Y'_p] \geq \left[\widehat{R}_p - \left\lfloor \frac{R_p}{2} \right\rfloor\right] \mu_T > 0;$$
$$(3) Y'_p = X'_p: \quad \mathrm{E}[Y_p - Y'_p] \geq \left[\widehat{R}_p - \left\lceil \frac{R_p}{2} \right\rceil - \frac{1}{2}\right] \mu_T > 0;$$

by assuming that $R_p > 1$. □

4.5 Analysis of AR Dynamic Load Balancing

In the AR algorithm, all remaining jobs on all processors are equitably redistributed among all processors right after P_1 finishes its jobs and becomes idle. At this time each processor P_i, $i = 2, \ldots, p$, has R_i remaining jobs and a remaining execution time X_i. One job is in progress with an expected completion time $\mu_T/2$ and $R_i - 1$ jobs are queued. R_i is known and not a random variable because the analysis is carried out at a given time. The total number of remaining jobs is $\sum_{i=1}^{p} R_i$. Let

$$b = \left(\sum_{i=1}^{p} R_i\right) \bmod p, \quad r = \lfloor \mathrm{M}(R) \rfloor = \mathrm{M}(R) - \frac{b}{p}, \quad \text{and} \quad \widehat{r} = r - \frac{1}{2}.$$

The new number of jobs that the AR algorithm assigns to processor P_i is

$$R'_i = \begin{cases} r, & \text{if } b = 0 \text{ and } i = 1, \ldots, p, \\ r, & \text{if } b \neq 0 \text{ and } i = 1, \ldots, p - b, \\ r + 1, & \text{if } b \neq 0 \text{ and } i = p - b + 1, \ldots, p. \end{cases}$$

Let X'_i denote the execution time of the jobs on P_i after the AR step. The expected value of X'_i is

$$\mathrm{E}[X'_i] = \begin{cases} r\,\mu_T, & \text{if } i = 1, \\ \widehat{r}\,\mu_T, & \text{if } i = 2, \ldots, p - b, \\ (\widehat{r} + 1)\,\mu_T, & \text{if } i = p - b + 1, \ldots, p. \end{cases} \tag{16}$$

Proposition 4 *The expected algebraic variance* (1) *of X' is smaller than the expected algebraic variance of X after an AR DLB step, that is,* $\mathrm{E}[\xi^2_{X'}] < \mathrm{E}[\xi^2_X]$, *assuming* $\mathrm{V}(\widehat{R}) > 1/4$.

Proof According to Lemma 4 the expected decrease in the algebraic variance of the execution times is proportional to the decrease in the algebraic variance of the modified number of jobs. The AR algorithm redistributes the number of jobs equitably, such that after the load balancing step the algebraic variance of the number of tasks is the smallest among all possible distributions. Therefore the AR load balancing algorithm decreases

the expected variability of execution times across processors by the maximum possible amount, and $\mathrm{E}[\xi^2_{X'}] < \mathrm{E}[\xi^2_X]$.

The algebraic variance after AR load balancing is

$$\mathrm{V}(\widehat{R}') = \frac{1}{p-1}\sum_{i=1}^{p}\left(\widehat{R}'_i - \mathrm{M}(\widehat{R}')\right)^2 = \frac{p\,(4b+1)-(2b+1)^2}{4\,p\,(p-1)} \le \frac{1}{4}$$

for $0 \le b \le p-1$. The decrease in the expected value of the algebraic variance of the compute times is $\mathrm{E}\left[\xi^2_X\right] - \mathrm{E}\left[\xi^2_{X'}\right] \ge \left(\mathrm{V}(\widehat{R}) - \frac{1}{4}\right)\,\mu_T^2$.

For the remaining part of the analysis consider the case where the mean number of jobs is large, $\mathrm{M}(R) \gg 1$. In this case $r+1 \approx r \approx \widehat{r}$, i.e., the jobs are nearly equally distributed to processors by the AR step. Moreover, the fact that one job has started on each of P_2 to P_p but not on P_1 has a negligible effect on the statistics of compute times (which are dominated by the large number of queued tasks). Therefore assume that $\mathrm{M}(R)$ is large, $b = 0$, and no jobs have started on any of the processors. The AR algorithm recursively returns to the initial circumstances of the previous AR step, but with a smaller number of jobs. The equal distribution of work and the CLT permit approximation of the compute times per processor $X'_1, \ldots, X'_p$ with i.i.d. Gaussian random variables.

Proposition 5 *If R_p is sufficiently large, the expected value of the largest idle time (2) is decreased after an AR DLB step, that is,* $\mathrm{E}\left[Y'_p - Y'_1\right] < \mathrm{E}\left[Y_p - Y_1\right]$.

Proof The maximum compute time before balancing is at least equal to the compute time on the processor with the largest number of remaining jobs (assumed to be P_p without loss of generality). This implies that $\mathrm{E}[Y_p] \ge \mathrm{E}[X_p] = R_p\,\mu_T$. Similarly, the minimum compute time is at most equal to the compute time on the processor with the smallest number of remaining jobs. Therefore $\mathrm{E}[Y_1] \le \mathrm{E}[X_1] = R_1\,\mu_T = 0$, $R_p\,\mu_T \le \mathrm{E}[Y_p - Y_1]$, and $\left(R_p - M(R)\right)\mu_T \le \mathrm{E}[Y_p - \eta_Y]$. The expected values of the greatest and the least order statistics in Gaussian samples can be accurately approximated using (6a)–(6b). Under the above simplifying assumptions ($b = 0$ and no processes have started) all the X_i are (approximately) i.i.d. normal random variables. From (6a), (6b), and (16),

$$\begin{aligned}\mathrm{E}[Y'_p] &= r\,\mu_T + \sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p}) + \mathrm{err}_p(p),\\ \mathrm{E}[Y'_1] &= r\,\mu_T - \sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p}) + \mathrm{err}_1(p).\end{aligned}$$

Assume that the relative approximation errors have an upper bound $\epsilon < 0.5$ for all $p \ge 20$:

$$\begin{aligned}\left|\mathrm{err_p}(p)\right| &\le \epsilon \cdot \left|r\,\mu_T + \sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p})\right|,\\ \left|\mathrm{err_1}(p)\right| &\le \epsilon \cdot \left|r\,\mu_T - \sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p})\right|,\end{aligned}$$

taking the relative errors with respect to the approximate values for convenience. Note that the results in [7] estimate $\epsilon \leq 0.04$. Consequently,

$$\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1] = 2\sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p}) + \mathrm{err_p}(p) - \mathrm{err}_1(p).$$

For bounded numbers of processors $p \leq p_{\max}$ the inverse function $\Phi^{-1}(0.5264^{1/p})$ is bounded by $\Phi^{-1}(0.5264^{1/p_{\max}}) = C_{\max} \approx 4.4$ for $p_{\max} = 1,000,000$. Therefore,

$$\begin{aligned}\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1] &\leq 2\,C_{\max}\,\sqrt{r}\,\sigma_T + \left|\mathrm{err_p}(p)\right| + |\mathrm{err}_1(p)| \\ &\leq 2\,(1+\epsilon)\,C_{\max}\,\sqrt{r}\,\sigma_T + 2\,\epsilon\,r\,\mu_T.\end{aligned}$$

The decrease in expected maximum idle time is at least

$$\begin{aligned}\mathrm{E}\left[Y_p - Y_1\right] - \mathrm{E}\left[Y'_p - Y'_1\right] &\geq \left(R_p - 2\,\epsilon\,r\right)\,\mu_T - 2\,(1+\epsilon)\,C_{\max}\,\sqrt{r}\,\sigma_T \\ &> (1-2\,\epsilon\,)\;R_p\,\mu_T - 2\,(1+\epsilon)\,C_{\max}\,\sqrt{R_p}\,\sigma_T \geq 0\end{aligned}$$

for $r < R_p$ and $R_p \geq \frac{4\,(1+\epsilon)^2\,C_{\max}^2}{(1-2\,\epsilon)^2}\left(\frac{\sigma_T}{\mu_T}\right)^2$.

This lower bound for R_p does not depend on p $(20 \leq p \leq p_{\max})$, but depends only on σ_T and μ_T. □

Proposition 6 *If $R_p > (1+\epsilon+g)\,r$ for some $g > 0$ and r is sufficiently large, the expected value of the average idle time* (3) *is decreased after an AR DLB step, that is,* $\mathrm{E}\left[Y'_p - \eta_X\right] < \mathrm{E}\left[Y_p - \eta_X\right]$.

Proof Before an AR load balancing step, since $\mathrm{E}[Y_p] \geq \mathrm{E}[X_p] = R_p\,\mu_T$ as before, the mean load imbalance is $\mathrm{E}\left[Y_p - \eta_X\right] \geq \left(R_p - r\right)\mu_T$.

After the AR step, and using ϵ from the proof of Proposition 5, the mean load imbalance becomes

$$\begin{aligned}&\mathrm{E}\left[Y'_p - \eta_X\right] \\ &\quad= r\,\mu_T + \sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p}) + \mathrm{err_p}(p) - r\mu_T \\ &\quad\leq (1+\epsilon)\,\left(r\,\mu_T + \sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p})\right) - r\mu_T \\ &\quad\leq \epsilon\,r\mu_T + (1+\epsilon)\,C_{\max}\,\sqrt{r}\,\sigma_T.\end{aligned}$$

Therefore, the difference after the AR step is

$$\begin{aligned}\mathrm{E}\left[Y_p - \eta_X\right] - \mathrm{E}\left[Y'_p - \eta_X\right] &\geq \left(R_p - r - \epsilon\,r\right)\,\mu_T - (1+\epsilon)\,C_{\max}\,\sqrt{r}\,\sigma_T \\ &> g\,r\,\mu_T - (1+\epsilon)\,C_{\max}\,\sqrt{r}\,\sigma_T.\end{aligned}$$

The expected mean idle time decreases if R_p is sufficiently large, when

$$R_p > (1+\epsilon+g)\, r \geq (1+\epsilon+g) \left(\frac{(1+\epsilon)\, C_{\max}\, \sigma_T}{g\, \mu_T} \right)^2 .$$

□

4.6 Analysis of RP Dynamic Load Balancing

Recall that P_1 is the first processor that finishes its tasks and becomes idle. In the RP algorithm, the idle processor sends requests to randomly chosen processors until a busy one is found. Assume, without loss of generality, that P_k is the busy processor that was chosen randomly. P_k has the load of R_k unfinished jobs. The RP load balancing step moves $\lfloor R_k/2 \rfloor$ jobs from the busy processor P_k to P_1. The loads of the processors other than P_1 and P_k are not changed. P_p has the highest load of R_p unfinished jobs as before. Let X'_i be the remaining compute time for processor P_i after the RP DLB step. From the above, $X'_i = X_i$ for $i = 2, \ldots, p$ and $i \neq k$. For the processors P_1 and P_k, the expected values of the compute times are $\mathrm{E}[X'_1] = R'_1\, \mu_T = \left\lfloor \frac{R_k}{2} \right\rfloor \mu_T$, and $\mathrm{E}[X'_k] = \left(R'_k - \frac{1}{2}\right) \mu_T = \left(\left\lceil \frac{R_k}{2} \right\rceil - \frac{1}{2} \right) \mu_T$.

The above expression accounts for the fact that P_k has one task in progress. Furthermore,

$$\mathrm{E}[X'_k] - \mathrm{E}[X'_1] = \left(R_k \bmod 2 - \frac{1}{2} \right) \mu_T .$$

The following propositions prove that each RP redistribution step decreases the level of load imbalance as measured by the metrics (1)–(3).

Proposition 7 *The expected value of the algebraic variance of the compute times per processor* (1) *decreases after a RP DLB step by*

$$\mathrm{E}[\xi_X^2] - \mathrm{E}[\xi_{X'}^2] = \frac{(\mathrm{M}(R))^2 - \mathrm{M}(R) + \mathrm{V}(R)}{2(p-1)}\, \mu_T^2 .$$

Proof Assume the probability that the P_k is randomly chosen as a donor processor is $1/(p-1)$. The average adjusted number of tasks per processor is the same before and after RP load balancing, $\mathrm{M}(\widehat{R}) = \mathrm{M}(\widehat{R}')$. The decrease in the algebraic variance of the adjusted number of tasks is

$$\begin{aligned} \mathrm{V}(\widehat{R}) - \mathrm{V}(\widehat{R}') = \frac{1}{p-1} \Big(& (\widehat{R}_1 - \mathrm{M}(\widehat{R}))^2 - (\widehat{R}'_1 - \mathrm{M}(\widehat{R}))^2 \\ & + (\widehat{R}_k - \mathrm{M}(\widehat{R}))^2 - (\widehat{R}'_k - \mathrm{M}(\widehat{R}))^2 \Big) = \frac{R_k\, (R_k - 1)}{2\, (p-1)} . \end{aligned}$$

Lemma 4 and the probability that P_k is randomly chosen as a donor processor provide the difference between the expected variances of compute times across processors before and after a RP DLB step

$$\begin{aligned}
\mathrm{E}[\xi_X^2] - \mathrm{E}[\xi_{X'}^2] &= \sum_{k=2}^{p} \frac{1}{p-1} \left(\frac{R_k\,(R_k-1)}{2\,(p-1)} \right) \mu_T^2 = \frac{\mu_T^2}{2(p-1)^2} \sum_{k=2}^{p} \left(R_k^2 - R_k \right) \\
&= \frac{\mu_T^2}{2(p-1)^2} \sum_{k=2}^{p} \left\{ (R_k - \mathrm{M}(R))^2 + (2\mathrm{M}(R) - 1)\, R_k - (\mathrm{M}(R))^2 \right\} \\
&= \frac{(\mathrm{M}(R)-1)^2 + (\mathrm{M}(R)-1) + \mathrm{V}(R)}{2(p-1)}\, \mu_T^2 > 0
\end{aligned}$$

for $\mathrm{M}(R) > 1$. □

Proposition 8 *The expected value of the largest idle time* (2) *is not increased after a RP DLB step, that is,* $\mathrm{E}\left[Y'_p - Y'_1 \right] \leq \mathrm{E}\left[Y_p - Y_1 \right]$.

Proof Before the RP load balancing step, the expected maximum imbalance is $\mathrm{E}[Y_p] - \mathrm{E}[Y_1] = \mathrm{E}[Y_p] \geq \mathrm{E}[X_p] = \widehat{R}_p\, \mu_T$.

After the RP load balancing step, the new expected maximum imbalance time is $\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]$. If $P_k = P_p$ has the highest load of R_p, the proof is the same as that for Proposition 2. Otherwise $1 < k < p$. Consider the random variables

$$\begin{aligned}
Z_{min} &= \min\{X_2, \ldots, X_{k-1}, X_{k+1}, \ldots, X_p\}, \\
Z_{max} &= \max\{X_2, \ldots, X_{k-1}, X_{k+1}, \ldots, X_p\} \;\leq\; Y_p.
\end{aligned}$$

The smallest and the largest order statistics after the RP load balancing step are $Y'_1 = \min\{X'_1, X'_k, Z_{min}\}$ and $Y'_p = \max\{X'_1, X'_k, Z_{max}\}$. Seven possible combinations of Y'_1 and Y'_p values are considered as in the proof of Proposition 2.

(1) $Y'_1 = Z_{min}$ and $Y'_p = Z_{max}$; (2) $Y'_1 = Z_{min}$ and $Y'_p = X'_k$;
(3) $Y'_1 = Z_{min}$ and $Y'_p = X'_1$; (4) $Y'_1 = X'_k$ and $Y'_p = Z_{max}$;
(5) $Y'_1 = X'_k$ and $Y'_p = X'_1$; (6) $Y'_1 = X'_1$ and $Y'_p = Z_{max}$;
(7) $Y'_1 = X'_1$ and $Y'_p = X'_k$.

In Case (1) the balanced times fall between Z_{min} and Z_{max}. That the expected maximum idle time reduction is greater than or equals to zero is proved the same as for Case (1) in Proposition 2. The reductions of expected maximum idle times for Cases (2) to (7) are shown by straightforward verification. Therefore, after a RP DLB step, the expected maximum time imbalance is always the same or reduced. Note that if $R_p \geq 2$ and $r \geq 1$, then the expected maximum time imbalance is always reduced. This condition is general in load balancing steps. □

Proposition 9 *The expected value of the average idle time* (3) *is not increased after a RP DLB step, that is,* $\mathrm{E}\left[Y'_p - \eta_{X'} \right] \leq \mathrm{E}\left[Y_p - \eta_X \right]$.

Proof The decrease in the expected average idle time is

$$\mathrm{E}\left[Y_p - \eta_X\right] - \mathrm{E}\left[Y'_p - \eta_{X'}\right] = \mathrm{E}\left[Y_p\right] - \mathrm{E}\left[Y'_p\right]$$

since $\eta_{X'} = \eta_X$.

Consider each of the possible values of Y'_p separately.

$$\begin{aligned}
&(1)\ Y'_p = Z_{max}: \ \mathrm{E}\left[Y_p\right] - \mathrm{E}\left[Y'_p\right] = \mathrm{E}\left[Y_p - Z_{max}\right] \geq 0;\\
&(2)\ Y'_p = X'_1: \quad \mathrm{E}\left[Y_p\right] - \mathrm{E}\left[Y'_p\right] \geq \left[R_p - \left\lfloor \frac{R_k}{2} \right\rfloor - \frac{1}{2}\right]\mu_T > 0;\\
&(3)\ Y'_p = X'_k: \quad \mathrm{E}\left[Y_p\right] - \mathrm{E}\left[Y'_p\right] \geq \left[R_p - \left\lceil \frac{R_k}{2} \right\rceil\right]\mu_T \geq 0.
\end{aligned}$$

In the first case $Y'_p = Z_{max}$, Z_{max} is the same as Y_p except when the donor processor P_k is randomly selected to be the most overloaded processor P_p. Therefore the expected value of the reduction in the average idle time is zero for most RP load balancing steps. □

4.7 Analysis of NR Dynamic Load Balancing

In the NR algorithm, the idle processor sends requests to neighbor processors to redistribute the remaining jobs on the neighbor processors and itself. Assume that the number of neighbor processors of the idle processor is $k-1$ and changes with different network topologies. Let P_1 be the idle processor that finishes its jobs with its neighbor processors $P_2, \ldots, P_k$. Before the NR load balancing step, each processor P_i, $i = 2, \ldots, k$, has R_i remaining jobs and a remaining execution time X_i. One job is in progress with an expected completion time $\mu_T/2$ and $R_i - 1$ jobs are queued. Let $b = \left(\sum_{i=1}^{k} R_i\right) \bmod k$, $\widetilde{\mathrm{M}}(R) = \frac{1}{k}\sum_{i=1}^{k} R_i$, $r = \left\lfloor \widetilde{\mathrm{M}}(R) \right\rfloor = \widetilde{\mathrm{M}}(R) - \frac{b}{k}$, and $\widehat{r} = r - \frac{1}{2}$.

The new number of jobs that the NR algorithm assigns to processor P_i is

$$R'_i = \begin{cases} r, & i = 1, \ldots, k-b,\\ r+1, & i = k-b+1, \ldots, k,\\ R_i, & i = k+1, \ldots, p. \end{cases}$$

Let X'_i denote the execution time of the jobs on P_i after the NR step. The expected value of X'_i is

$$\mathrm{E}[X'_i] = \begin{cases} r\,\mu_T, & \text{if } i = 1,\\ \widehat{r}\,\mu_T, & \text{if } i = 2, \ldots, k-b,\\ (\widehat{r}+1)\,\mu_T, & \text{if } i = k-b+1, \ldots, k.\\ \widehat{R}_i\,\mu_T, & \text{if } i = k+1, \ldots, p. \end{cases}$$

Define $\widetilde{\mathrm{M}}(\widehat{R}) = \frac{1}{k}\sum_{i=1}^{k} \widehat{R}_i$, $\widetilde{\mathrm{V}}(\widehat{R}) = \frac{1}{k-1}\sum_{i=1}^{k} \left(\widehat{R}_i - \widetilde{\mathrm{M}}(\widehat{R})\right)^2$.

Proposition 10 *The expected algebraic variance* (1) *of* X' *is smaller than the expected algebraic variance of* X *after an NR DLB step, that is,* $\mathrm{E}[\xi^2_{X'}] < \mathrm{E}[\xi^2_X]$, *assuming* $\widetilde{\mathrm{V}}(\widehat{R}) > 1/4$.

Proof Since $R'_i = R_i$ for $i = k+1, \ldots, p$,

$$\mathrm{V}(\widehat{R}) - \mathrm{V}(\widehat{R}') = \frac{1}{p-1}\left\{\sum_{i=1}^{k}\left(\widehat{R}_i\right)^2 - \sum_{i=1}^{k}\left(\widehat{R}'_i\right)^2\right\} = \frac{k-1}{p-1}\left(\widetilde{\mathrm{V}}(\widehat{R}) - \widetilde{\mathrm{V}}(\widehat{R}')\right).$$

Proposition 4 provides the algebraic variance of the modified number of jobs after the NR load balancing step, since it is the same as the AR load balancing step for P_i where $i = 1, \ldots, k$.

$$\widetilde{\mathrm{V}}(\widehat{R}') = \frac{1}{k-1}\sum_{i=1}^{k}\left(\widehat{R}'_i - \widetilde{\mathrm{M}}(\widehat{R}')\right)^2 = \frac{k\,(4b+1) - (2b+1)^2}{4\,k\,(k-1)} \leq \frac{1}{4}.$$

for $0 \leq b \leq k-1$. Finally the decrease in the expected value of the algebraic variance of the compute times is

$$\mathrm{E}\left[\xi^2_X\right] - \mathrm{E}\left[\xi^2_{X'}\right] \geq \frac{k-1}{p-1}\left(\widetilde{\mathrm{V}}(\widehat{R}) - \frac{1}{4}\right)\mu_T^2.$$

□

Proposition 11 *The expected value of the largest idle time* (2) *is monotonically decreased after a NR DLB step, that is,* $\mathrm{E}\left[Y'_p - Y'_1\right] \leq \mathrm{E}\left[Y_p - Y_1\right]$.

Proof Before the NR load balancing step the expected maximum imbalance is

$$\mathrm{E}[Y_p] - \mathrm{E}[Y_1] = \mathrm{E}[Y_p] \geq \mathrm{E}[X_p] = \widehat{R}_p\,\mu_T.$$

After the NR load balancing step, the new expected maximum imbalance time is $\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]$. Consider the random variables

$$Z_{min} = \min\{X'_1, \ldots, X'_k\},\ Z_{max} = \max\{X'_1, \ldots, X'_k\} \leq \max\{X_1, \ldots, X_k\} \leq Y_p\,,$$
$$W_{min} = \min\{X'_{k+1}, \ldots, X'_p\} \geq Y_1 = 0\,,\ W_{max} = \max\{X'_{k+1}, \ldots, X'_p\} \leq Y_p.$$

The smallest and the largest order statistics after NR balancing are $Y'_1 = \min\{Z_{min}, W_{min}\}$ and $Y'_p = \max\{Z_{max}, W_{max}\}$. There are four possible combinations of Y'_1 and Y'_p values:

(1) $Y'_1 = Z_{min}$ and $Y'_p = W_{max}$; (2) $Y'_1 = W_{min}$ and $Y'_p = W_{max}$;
(3) $Y'_1 = Z_{min}$ and $Y'_p = Z_{max}$; (4) $Y'_1 = W_{min}$ and $Y'_p = Z_{max}$.

$$
\begin{aligned}
\text{Case (1)} &: \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} \\
&= \{\mathrm{E}[Y_p] - \mathrm{E}[W_{max}]\} + \mathrm{E}[Z_{min}] \geq \mathrm{E}[Z_{min}] \geq 0. \\
\text{Case (2)} &: \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} \\
&= \{\mathrm{E}[Y_p] - \mathrm{E}[W_{max}]\} + \mathrm{E}[W_{min}] \ \geq \ 0. \\
\text{Case (3)} &: \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} \\
&= \{\mathrm{E}[Y_p] - \mathrm{E}[Z_{max}]\} + \mathrm{E}[Z_{min}] \ \geq \ 0. \\
\text{Case (4)} &: \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\} \\
&= \{\mathrm{E}[Y_p] - \mathrm{E}[Z_{max}]\} + \mathrm{E}[W_{min}] \ \geq \ 0.
\end{aligned}
$$

□

Proposition 12 *The expected value of the average idle time* (3) *does not increase after a NR load balancing step, that is,* $\mathrm{E}\left[Y'_p - \eta_{X'}\right] \leq \mathrm{E}\left[Y_p - \eta_X\right]$.

Proof The decrease in the expected average idle time is (since $\eta_{X'} = \eta_X$ by Lemma 1)

$$\mathrm{E}\left[Y_p - \eta_X\right] - \mathrm{E}\left[Y'_p - \eta_{X'}\right] = \mathrm{E}\left[Y_p\right] - \mathrm{E}\left[Y'_p\right].$$

From the proof of Proposition 11,

$$Y'_p = \max\{Z_{max}, W_{max}\} \leq Y_p.$$

Therefore, $\mathrm{E}\left[Y_p\right] - \mathrm{E}\left[Y'_p\right] \geq 0$. □

5 Theoretical and Experimental Results

This section provides theoretical and experimental load balancing results with the budding yeast cell cycle model. To evaluate the four load balancing algorithms, the ensemble of simulations is executed on Virginia Tech's System X supercomputer [33]. The supercomputer has 1,100 Apple PowerMac G5 nodes, with dual 2.3 GHz PowerPC 970FX processors and 4GB memory.

5.1 Stochastic Simulation of the Budding Yeast Cell Cycle Model

The cycle of cell growth, DNA synthesis, mitosis, and cell division is the fundamental process by which cells grow, develop, and reproduce. The molecular machinery of eukaryotic cell cycle control is known in more detail for budding yeast, *Saccharomyces cerevisiae*, than for any other organism. Therefore, the unicellular budding yeast is an excellent organism for which to study cell cycle regulation.

We have implemented a stochastic model for the budding yeast cell cycle [1,36] based on the original model of Chen et al. [8]. Gillespie's SSA [14] is executed on the cell cycle model. To accurately mimic the experimental protocol, we choose cells

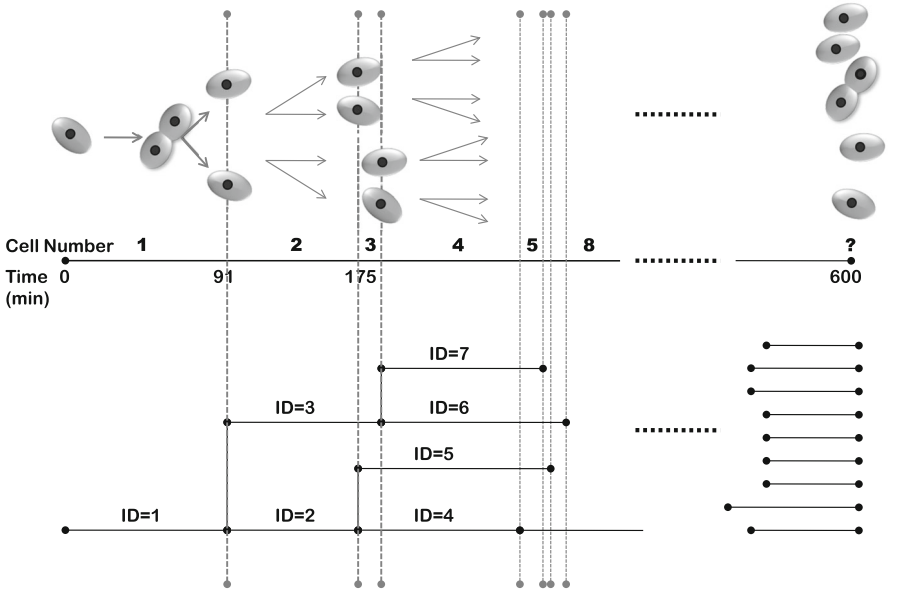

Fig. 4 Multistage cell cycle tracking diagram. ID is the cell identification tag. Cell modeling simulations should be executed beginning at each cell emergence time

from a specific distribution of initial conditions, and simulate all of their progeny. Existing stochastic simulators based on the Gillespie's SSA treat one system with one initial molecular state vector. To simulate all of the progeny, whose initial states are different, multicycle cell lineage tracking is needed, as illustrated in Fig. 4. Biologists are interested in the number of cells in existence at a specific final time. The algorithm for the multistage cell cycle implementation is described in detail in [2].

5.2 Numerical Evaluation of Static Distribution

To assess how well the theoretical estimates of load imbalance metrics agree with the simulation results, consider the case with $n = 1,000$ cell cycle simulations distributed evenly across $p = 25$ processors, which results in $R = 40$ tasks per processor. To evaluate probabilistic measures the expected maximum CPU time $\mathrm{E}[Y_p]$ and minimum CPU time $\mathrm{E}[Y_1]$ can be calculated in two ways: the integral method (14a)–(14b) and the approximation method (6a)–(6b). $\mathrm{E}[Y_p] = 20,973$ and $\mathrm{E}[Y_1] = 18,075$ calculated from the integral method are similar to the approximation method results of $\mathrm{E}[Y_p] = 20,965$ and $\mathrm{E}[Y_1] = 18,083$. Results from both methods match the experimental results in Table 1.

Probabilistic measures (1)–(3) of load imbalance are the root expected algebraic variance of times across the processors,

$$\sqrt{\mathrm{E}\left[\xi_X^2\right]} = \sqrt{\frac{p}{p-1} \cdot R \cdot \sigma_T^2} \approx 752.65 \text{ seconds},$$

Table 1 Average, maximum, minimum, RAV (square root of the algebraic variance) of compute times, maximum idle time, and average (percentage) idle time for wild-type cell simulations

Metrics	Static	MD	AR	RP	NR
1,000 Runs (25 processors)					
Avg comp. time	19,524	19,362	19,277	19,515	19,578
Max comp. time	20,795	19,781	19,709	19,836	19,931
Min comp. time	18,055	19,084	19,033	19,254	19,175
RAV comp. time	680	195	172	150	210
Max idle time	2,740	697	676	582	756
Avg idle time	1,271	419	432	321	352
Idle time (%)	6.5	2.2	2.2	1.7	1.8
Theoretical $\mathrm{E}[\sqrt{\xi_X^2}]$	753	240	198	220	241
10,000 Runs (100 processors)					
Avg comp. time	47,880	47,778	48,039	47,991	48,020
Max comp. time	51,050	48,289	48,413	48,412	48,556
Min comp. time	44,431	47,354	47,802	47,725	47,643
RAV comp. time	1,272	196	154	187	227
Max idle time	6,619	935	611	687	914
Avg idle time	3,170	511	374	421	536
Idle time (%)	6.6	1.1	0.8	0.9	1.1
Theoretical $\mathrm{E}[\sqrt{\xi_X^2}]$	1,176	229	198	297	378

The static and the four explored load balancing approaches are compared by results from both a small and a large ensemble. Units are seconds

the expected worst case load imbalance,

$$\mathrm{E}[Y_p] - \mathrm{E}[Y_1] \approx 2,898 \text{ seconds},$$

and the expected idle time per processor,

$$\mathrm{E}\left[Y_p - \eta_X\right] \approx 1,449 \text{ seconds}.$$

In the simulation experiment based on 1,000 simulations with fa static distribution over 25 processors, the root algebraic variance of CPU times is 679.83 s, the maximum load imbalance $Y_p - Y_1$ is 2,740.42 s, and the average CPU idle time $(1/p)\sum_{i=1}^{p}(Y_p - X_i) = 1,270.75$ s. The theoretical probabilistic measures of load imbalance are consistent with the simulation experiment values.

5.3 Numerical Evaluation of Theoretical Analysis for the Load Balancing Algorithms

In this section, the approximations employed in the theoretical analysis of the four different dynamic load balancing algorithms are compared to the experimental results numerically. Figure 5a–d compare the theoretical root expected algebraic variance of

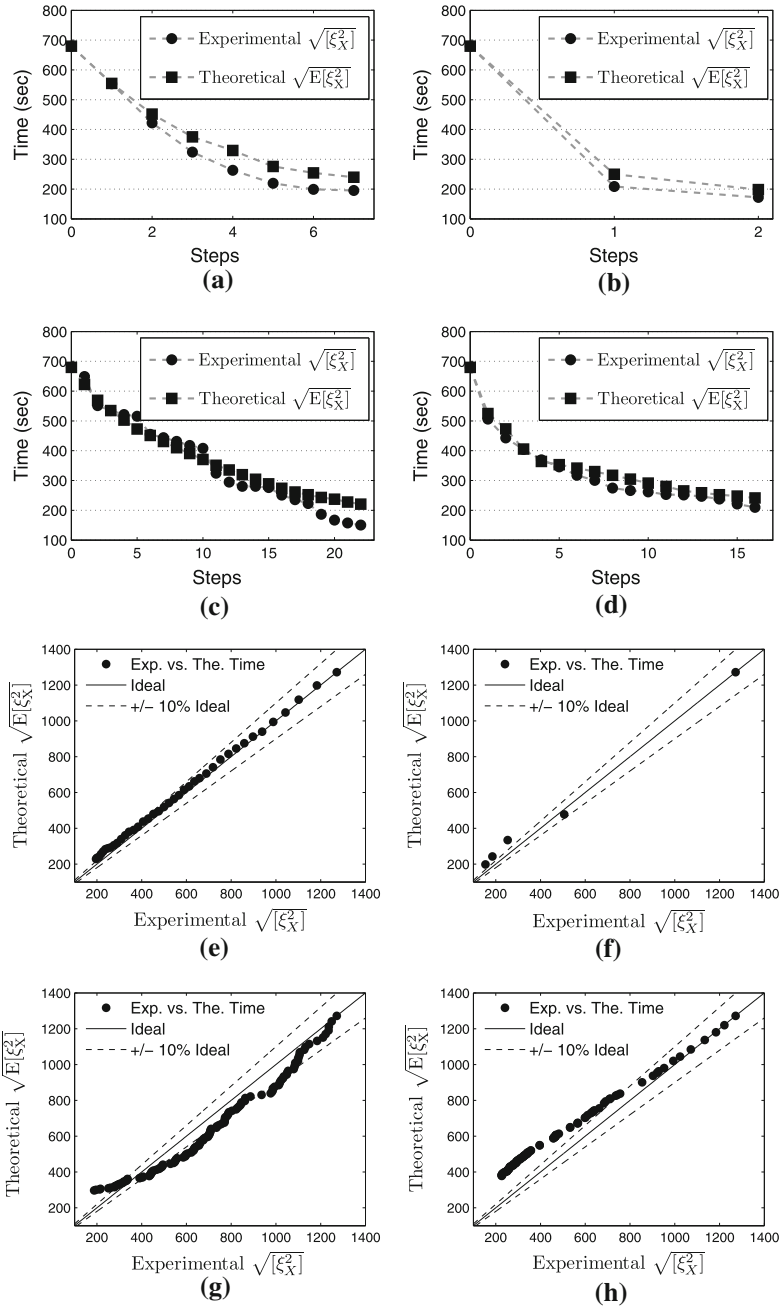

Fig. 5 Numerical comparison of the experimental RAV to the theoretical root expected algebraic variance of compute times across the processors for the four load balancing algorithms. 1,000 runs with 25 processors for **a**–**d** and 10,000 runs with 100 processors for **e**–**h**, **a** MD load balancing, **b** AR load balancing, **c** RP load balancing, **d** NR load balancing, **e** MD load balancing, **f** AR load balancing, **g** RP load balancing, **h** NR load balancing

compute times across the processors for each load balancing step to the experimental square root of the algebraic variance (RAV) with $n = 1,000$ tasks on $p = 25$ processors. To investigate in the case of many tasks on many processors, the theoretical and experimental results of $n = 10,000$ tasks on $p = 100$ processors are considered in Fig. 5e–h.

The numerical reduction of the expected algebraic variance of the compute times across the processors before and after a load balancing step is quantified in Propositions 1, 4, 7, and 10 for each load balancing algorithm. In the MD analysis, Proposition 1 provides that the numerical root expected algebraic variance of the compute times across the processors after a load balancing step that is

$$\sqrt{\mathrm{E}[\xi_{X'}^2]} = \sqrt{\mathrm{E}[\xi_X^2] - \frac{R_p\,(R_p - 1)}{2\,(p-1)}\,\mu_T^2}.$$

In the AR analysis, Proposition 4 provides that

$$\sqrt{\mathrm{E}[\xi_{X'}^2]} = \sqrt{\mathrm{E}[\xi_X^2] - \left(\mathrm{V}(\widehat{R}) - \mathrm{V}(\widehat{R}')\right)\mu_T^2}$$

where $\mathrm{V}(\widehat{R}') = \frac{p\,(4b+1)-(2b+1)^2}{4\,p\,(p-1)}$.

In the RP analysis, Proposition 7 provides that

$$\sqrt{\mathrm{E}[\xi_{X'}^2]} = \sqrt{\mathrm{E}[\xi_X^2] - \frac{(\mathrm{M}(R))^2 - \mathrm{M}(R) + \mathrm{V}(R)}{2(p-1)}\mu_T^2}.$$

In the NR analysis, Proposition 10 provides that

$$\sqrt{\mathrm{E}[\xi_{X'}^2]} = \sqrt{\mathrm{E}[\xi_X^2] - \frac{k-1}{p-1}\left(\widetilde{\mathrm{V}}(\widehat{R}) - \widetilde{\mathrm{V}}(\widehat{R}')\right)\mu_T^2}$$

where $\widetilde{\mathrm{V}}(\widehat{R}') = \frac{k\,(4b+1)-(2b+1)^2}{4\,k\,(k-1)}$. Figure 5 shows that the variance decreases consistently on each iteration as predicted by the theory for all load balancing cases. Table 1 shows the final root expected algebraic variance of the compute times versus experimental root of the algebraic variance across the processors. As expected, the theoretical values are larger than experimental results since the theory provides upper bounds for the metric (1).

5.4 Load Balancing Results for Wild-Type Yeast

This section describes load balancing results for the wild-type and a prototype mutant budding yeast cell cycle models. 1,000 simulations with 25 processors and 10,000 simulations with 100 processors are executed for these experiments. With the static distribution of the wild-type simulations, the variance of the CPU times is not huge because wild-type cells divide in a relatively regular fashion. The simulation time is

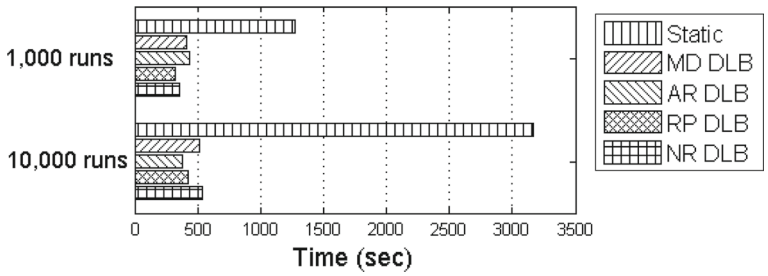

Fig. 6 The average idle CPU times comparison for the static distribution and the final step of the load balancing methods

just affected by the stochastic nature of the SSA. Nevertheless, the four DLB methods reduce the wall clock time compared to the static method; the differences are approximately 1,000 s (4.9 % of static distribution wall clock time) for 1,000 runs with 25 processors, and 2,800 s (5.4 % of static distribution wall clock time) for 10,000 runs with 100 processors. Table 1 demonstrates the efficiency of the four DLB algorithms clearly.

For the static load balancing case of the prototype mutant simulations, the variance of compute times is huge because of the characteristics of mutant simulations. The DLB algorithms reduce the wall clock times by approximately 26 % for 1,000 runs with 25 processors, and by approximately 21 % for 10,000 runs with 100 processors. The four DLB algorithms lead to greater improvements for the mutant than for the wild-type simulations. Appendix C shows detailed experimental results of the mutant simulations.

Figure 6 compares the average idle CPU times for the static and four DLB algorithms. For the wild-type simulations, the DLB algorithms have eliminated approximately two thirds of the idle time for 25 processors (from 6.5 % of the total CPU time down to 2 % of the total CPU time), and roughly 85 % for the 100 processor experiment (from 7 % of the total CPU time down to 1 % of the total CPU time).

The communication time for the load balancing methods should be considered. The total communication times for the four DLB algorithms are approximately 0.2 s for 1,000 runs with 25 processors and 1.0 s for 10,000 runs with 100 processors. Therefore, the total communication time for the DLB is negligible compared to elapsed wall clock time. The centralized and decentralized DLB algorithms have similar performance for these simulations without considering scalability.

5.5 Load Balancing Results for Mutant Yeast

This section presents experimental results for a prototype budding yeast mutant cell cycle model. For the mutant strain considered, the initial cell might never divide at all or it might divide several times and then cease division [8]. Therefore, the CPU time to simulate such a mutant cell varies, even if the end time of the simulation is fixed. For these simulations, the four dynamic load balancing algorithms show huge advantages in CPU utilization.

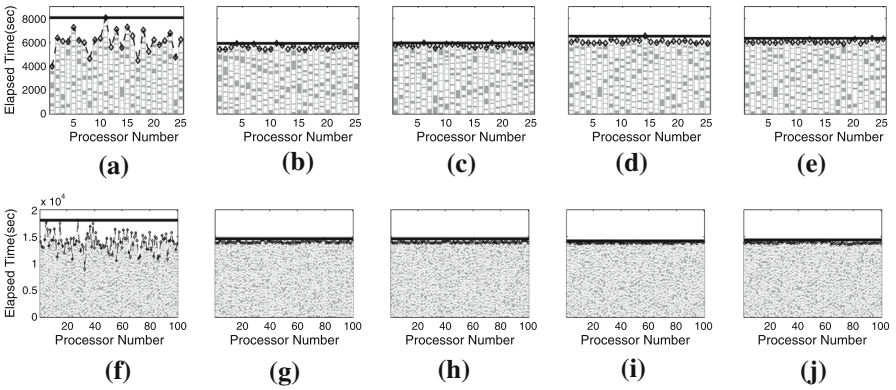

Fig. 7 Elapsed compute times per processor (*diamond marker*) and wall clock time (*solid line*) of prototype mutant multistage cell cycle simulations. 1,000 runs with 25 processors for **a**–**e** and 10,000 runs with 100 processors for **f**–**j**. *Small grey rectangular height* represents each job time for the processor, **a** Static **b** MD DLB, **c** AR DLB, **d** RP DLB, **e** NR DLB, **f** Static, **g** MD DLB, **h** AR DLB, **i** RP DLB, **j** NR DLB

Figure 7 shows the overall wall clock times per processor. Figure 7a–e show results for 1,000 runs with 25 processors and Fig. 7f–j show results for 10,000 runs with 100 processors. For the static load balancing case, the variance of compute times is huge because of the characteristics of mutant simulations. The DLB algorithms reduce the wall clock times by approximately 26 % for 1,000 runs with 25 processors, and by approximately 21 % for 10,000 runs with 100 processors. The four DLB algorithms lead to greater improvements for the mutant than for the wild-type simulation.

Table 2 also shows the improved efficiency of the four DLB algorithms compared to a static method. Statements similar to those for Table 1 can be made about Table 2, but the differences for mutant simulation are considerably more pronounced than for wild-type simulation. Average processor idle time was reduced by 85 % or more for each dynamic algorithm and on each ensemble (from 30 % of the total CPU time down to only 4 % of the total CPU time).

6 Conclusions and Future Work

This paper introduces a new probabilistic framework to analyze the effectiveness of load balancing strategies in the context of large ensembles of stochastic simulations. Ensemble simulations are employed to estimate the statistics of possible future states of the system, and are widely used in important applications such as climate change and biological modeling. The present work is motivated by stochastic cell cycle modeling, but the proposed analysis framework can be directly applied to any ensemble simulation where many tasks are mapped onto each processor, and where the task compute times vary considerably.

The analysis assumes only that the compute times of individual tasks can be modeled as independent identically distributed random variables. This is a natural assumption for an ensemble computation, where the same model is run repeatedly with different initial conditions and parameter values. No assumption is made about the shape of the

Table 2 Average, maximum, minimum, RAV (square root of the algebraic variance) of compute times, maximum idle time, and average (percentage) idle time for mutant cell simulations

Metrics	Static	MD	AR	RP	NR
1,000 Runs (25 processors)					
Avg comp. time	6,079	5,612	5,740	6,090	6,044
Max comp. time	8,054	5,922	5,965	6,387	6,333
Min comp. time	3,995	5,417	5,554	5,921	5,865
RAV comp. time	943	165	118	125	150
Max idle time	4,059	505	411	466	468
Avg idle time	1,975	310	225	297	289
Idle time (%)	32.5	5.5	3.9	4.9	4.8
10,000 Runs (100 processors)					
Avg comp. time	13,921	14,042	13,990	13,911	13,927
Max comp. time	18,038	14,617	14,606	14,371	14,466
Min comp. time	8,950	13,801	13,815	13,767	13,545
RAV comp. time	1,695	193	164	137	229
Max idle time	9,088	815	791	604	921
Avg idle time	4,117	575	616	460	539
Idle time (%)	29.6	4.1	4.4	3.3	3.9

The static and the four explored load balancing approaches are compared by results from both a small and a large ensemble. Units are seconds

underlying probability density; therefore the analysis is widely applicable. The level of load imbalance, as given by well defined metrics, is also a random variable. The analysis focuses on determining the decrease in the expected value of load imbalance after each work redistribution step. The analysis is applied to the four dynamic load balancing strategies. The analysis reveals that the expected level of load imbalance is monotonically decreased after one step of each of the algorithms.

Numerical results support the theoretical analysis. On an ensemble of budding yeast cell cycle simulations, compute times required to simulate each cell cycle progression using Gillespie's algorithm are inherently variable due to the stochastic nature of the model. Dynamic load balancing reduces the total compute (wall clock) times by about 5 % for ensembles of wild type cells, and by about 25 % for ensembles of mutant cells. Average processor idle time is reduced by 85 % or more for ensembles of mutant cells, which have widely varying running times.

An interesting direction that will be investigated in the future is to use the proposed analysis framework to design improved dynamic load balancing algorithms (and perhaps even optimal algorithms with respect to certain load balancing metrics). Scalability, not investigated here, will also be analyzed in the future. Since the centralized load balancing algorithms are not expected to scale well, the results of our analysis showing global improvements by the local load balancing algorithms in especially significant. Finally, a challenging problem is to analyze load balancing for large ensemble runs with different models, where the i.i.d. assumption does not hold.

Acknowledgments The authors thank the two anonymous reviewers whose comments helped improve this work. This work is supported in part by awards NIGMS/NIH 5 R01 GM078989, AFOSR FA9550-09-1-0153, NSF DMS-0540675, NSF CCF-0916493, NSF OCI-0904397, NSF DMS-1225160, and NSF CCF-0953590.

References

1. Ahn, T.-H., Watson, L., Cao, Y., Shaffer, C., Baumann, W.: Cell cycle modeling for budding yeast with stochastic simulation algorithms. Comput. Model. Eng. Sci. **51**(1), 27–52 (2009)
2. Ball, D., Ahn, T.-H., Wang, P., Chen, K., Cao, Y., Tyson, J., Peccoud, J., Baumann, W.: Stochastic exit from mitosis in budding yeast: model predictions and experimental observations. Cell Cycle **10**, 999–1099 (2011)
3. Bast, H.: Dynamic scheduling with incomplete information. In: Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '98), pp. 182–191. ACM, New York, NY, USA (1998)
4. Bast, H.: Provably Optimal Scheduling of Similar Tasks. Ph.D. thesis, Universitat des Saarlandes (2000)
5. Bertsekas, D., Tsitsiklis, J.: Parallel and Distributed Computation: Numerical Methods. Prentice-Hall, Upper Saddle River (1989)
6. Blumofe, R., Leiserson, C.: Scheduling multithreaded computations by work stealing. In: Proceedings of Annunal Symposyum on Foundations of Computer Science, pp. 356–368 (1994)
7. Chen, C.C., Tyler, C.: Accurate approximation to the extreme order statistics of Gaussian samples. Commun. Stat. Simul. Comput. **28**(1), 177–188 (1999)
8. Chen, K., Calzone, L., Csikasz-Nagy, A., Cross, F., Novak, B., Tyson, J.: Integrative analysis of cell cycle control in budding yeast. Mol. Biol. Cell **15**(8), 3841–3862 (2004)
9. Chu, W., Holloway, L., Lan, M.T., Efe, K.: Task allocation in distributed data processing. Computer **13**(11), 57–69 (1980)
10. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. J. Parallel Distrib. Comput. **7**, 279–301 (1989)
11. David, H., Nagaraja, H.: Order Statistics, 2nd edn. Wiley, Hoboken (2003)
12. Dijkstra, E., Scholten, C.: Termination detection for diffusing computations. Inf. Process. Lett. **11**(1), 1–4 (1980)
13. Flynn, L., Hummel, S.: The Mathematical Foundations of the Factoring Scheduling Method. Tech. rep., IBM Research, Report RC18462 (1992)
14. Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
15. Grama, A., Karypis, G., Kumar, V., Gupta, A.: Introduction to Parallel Computing, 2nd edn. Addison-Wesley, Boston (2002)
16. Hagerup, T.: Allocating independent tasks to parallel processors: An experimental study. J. Parallel Distrib. Comput. **47**(2), 185–197 (1997). http://www.sciencedirect.com/science/article/pii/S0743731597914118
17. Hillis, W.: The Connection Machine. MIT Press, Cambridge (1986)
18. Hummel, S., Schonberg, E., Flynn, L.: Factoring: a practical and robust method for scheduling parallel loops. Commun. ACM **35**(8), 90–101 (1992)
19. Iqbal, M., Saltz, J., Bokhari, S.: A comparative analysis of static and dynamic load balancing strategies. ACM Perform. Eval. Revis. **11**(1), 1040–1047 (1985)
20. Jacob, J., Lee, S.Y.: Task spreading and shrinking on a network of workstations with various edge classes. In: Proceedings of the 1996 International Conference on Parallel Processing, vol. 3, pp. 174–181 (1996)
21. Karp, R., Zhang, Y.: Randomized parallel algorithms for backtrack search and branch-and-bound computation. J. ACM **40**, 765–789 (1993)
22. Kruskal, C., Weiss, A.: Allocating independent subtasks on parallel processors. IEEE Trans. Softw. Eng. **SE-11**(10), 1001–1016 (1985)
23. Lester, B.: The Art of Parallel Programming. Prentice-Hall, Upper Saddle River (1993)
24. Lucco, S.: A dynamic scheduling method for irregular parallel programs. SIGPLAN Not. **27**(7), 200–211 (1992)

25. McAdams, H., Arkin, A.: Stochastic mechanisms in gene expression. Proc. Natl. Acad. Sci. **94**, 814–819 (1997)
26. Murphy, J., Sexton, D., Barnett, D., Jones, G., Webb, M., Collins, M., Stainforth, D.: Quantification of modelling uncertainties in a large ensemble of climate change simulations. Nature **430**, 768–772 (2004)
27. Polychronopoulos, C., Kuck, D.: Guided self-scheduling: a practical scheduling scheme for parallel supercomputers. IEEE Trans. Comput. **36**, 1425–1439 (1987)
28. Powley, C., Ferguson, C., Korf, R.: Depth-first heuristic search on a SIMD machine. Artif. Intell. **60**(2), 199–242 (1993)
29. Randles, M., Lamb, D., Taleb-Bendiab, A.: A comparative study into distributed load balancing algorithms for cloud computing. In: 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp. 551–556 (2010)
30. Ren, X., Lin, R., Zou, H.: A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast. In: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 220–224 (2011)
31. Rice, J.: Mathematical Statistics and Data Analysis, 3rd edn. Duxbury Press, Belmont (2001)
32. Rudolph, L., Slivkin-Allalouf, M., Upfal, E.: A simple load balancing scheme for task allocation in parallel machines. In: Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '91), pp. 237–245. ACM, New York, NY, USA (1991)
33. System X Supercomputer. http://www.arc.vt.edu/arc/SystemX/
34. Shavit, N., Francez, N.: A new approach to detection of locally indicative stability. In: Proceedings of the 13th International Colloquium on Automata, Languages and Programming (ICALP '86), pp. 344–358. Springer, London (1986)
35. Trivedi, K.: Probability and Statistics with Reliability, Queueing, and Computer Science Applications, 2nd edn. Wiley, Hoboken (2001)
36. Wang, P., Randhawa, R., Shaffer, C., Cao, Y., Baumann, W.: Converting macromolecular regulatory models from deterministic to stochastic formulation. In: Proceedings of the 2008 Spring Simulation Multiconference (SpringSim'08), High Performance Computing Symposium (HPC-2008), pp. 385–392. Society for Computer Simulation International, San Diego, CA, USA (2008)
37. Wilkinson, B., Allen, M.: Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, 2nd edn. Prentice-Hall, Upper Saddle River (2004)
38. Xu, C.Z., Lau, F.C.M.: Analysis of the generalized dimension exchange method for dynamic load balancing. J. Parallel Distrib. Comput. **16**(4), 385–393 (1992)
39. Zhang, Z., Zhang, X.: A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In: 2010 2nd International Conference on Industrial Mechatronics and Automation (ICIMA), vol. 2, pp. 240–243 (2010)