Parallel Load Balancing Strategies for Ensembles of Stochastic Biochemical Simulations

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

Abstract—The evolution of biochemical systems where some chemical species are present with only a small number of molecules, is strongly influenced by discrete and stochastic effects that cannot be accurately captured by continuous and deterministic models. The budding yeast cell cycle provides an excellent example of the need to account for stochastic effects in biochemical reactions. To obtain statistics of the cell cycle progression, a stochastic simulation algorithm must be run thousands of times with different initial conditions and parameter values. In order to manage the computational expense involved, the large ensemble of runs needs to be executed in parallel. The CPU time for each individual task is unknown before execution, so a simple strategy of assigning an equal number of tasks per processor can lead to considerable work imbalances and loss of parallel efficiency. Moreover, deterministic analysis approaches are ill suited for assessing the effectiveness of load balancing algorithms in this context. Biological models often require stochastic simulation. Since generating an ensemble of simulation results is computationally intensive, it is important to make efficient use of computer resources. This paper presents a new probabilistic framework to analyze the performance of dynamic load balancing algorithms when applied to large ensembles of stochastic biochemical simulations. Two particular load balancing strategies (point-to-point and all-redistribution) are discussed in detail. Simulation results with a stochastic budding yeast cell cycle model confirm the theoretical analysis. While this work is motivated by cell cycle modeling, the proposed analysis framework is general and can be directly applied to any ensemble simulation of biological systems where many tasks are mapped onto each processor, and where the individual compute times vary considerably among tasks.

Index Terms—Stochastic simulation algorithm (SSA), parallel computing, load balancing, cell cycle, budding yeast.

1 Introduction

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 [1]. Systems in which the stochastic effects are important must be described statistically. Large ensembles of simulations are needed to capture multiple evolutions of the system [2], [3], and to sample the probability density of all possible future states.

The motivating system for this work is the growth and division of eukaryotic cells [4]. A deterministic model [5] of the cell cycle for budding yeast, *Saccharomyces cerevisiae*, was converted to a set of chemical reaction equations and simulated stochastically [6] using Gillespie's stochastic simulation algorithm [7]. The statistics of interest are the time between birth and division of the cells, the probability that a single cell will grow to a colony of a specified size, and the average growth rate of the cells in culture. To accurately model the real system, simulations of a single cell, chosen from a specified distribution of initial conditions, and all of its progeny are run until a specified end time by the multistage cell tracking implementation. By running many such simulations the desired statistics can be computed.

The obvious way to speed up the overall simulation is to distribute the initial cells to different processors and run the simulations in parallel. For wild-type cells, load balancing among processors is not a huge issue. Wild-type cells divide in a relatively regular fashion so each simulation starting from a single cell and running to a specified end time will take roughly the same CPU time. There will be variations in total CPU time due to the stochastic nature of the chemical reactions and the effect of this on the division time and total number of progeny, but great differences are not expected to arise.

The load imbalance for mutant cells, however, can be quite significant. Mutant cells (e.g., cells deficient in certain genes) are often studied to validate the accuracy of models or to gain greater understanding of the functioning of the system. For certain types of mutations, the cells can behave erratically. A cell may never divide, it may divide a number of times but ultimately form a colony of fixed size where all of

- *T.-H. Ahn, A. Sandu, L.T. Watson, C.A. Shaffer, and Y. Cao are with the Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA. E-mail: {thahn, ltw, sandu, shaffer, ycao}@cs.vt.edu*
- *L.T. Watson is also with the Department of Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA.*
- *W.T. Baumann is with the Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA. E-mail: baumann@vt.edu*

the cells have exited the cell cycle, or it may divide endlessly and form a growing colony [5], [8]. The CPU time required to simulate these different situations, even if the end time of the simulations is fixed, can vary over a wide range. For these simulations, load balancing among processors is necessary to avoid wasting computing resources and power.

There is a large body of research available in the literature on static and dynamic scheduling techniques for load balancing [9], [10], [11], [12], [13]. The novelty of the work presented in this paper consists of a new framework for analyzing load balancing algorithms when applied to 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 framework is used to analyze two dynamic load balancing strategies: point-to-point (P2P) and all-redistribution (AR). Both strategies are centralized schemes [14], [15]. In the P2P algorithm, the processor that first finishes its work receives new tasks from the most overloaded processor. In the AR algorithm, when one worker becomes idle, the master redistributes all remaining jobs evenly among all processors. Both techniques use the termination detection approach of Bertsekas and Tsitsiklis [11] based on request and acknowledgement messages. The probabilistic analysis reveals that both load balancing strategies are effective for moderate parallelism; scalability is not investigated here.

Despite the apparent complexity of the analysis, the two dynamic load balancing strategies described here are easy to implement in place of a static allocation scheme. The resulting savings in computation time make them extremely useful tools for the practicing computational biologist.

The paper is organized as follows. Section 2 describes the cell cycle model. The two load balancing algorithms are presented in Section 3. Section 4 explains the analysis framework, and Section 5 contains the probabilistic analysis of the load balancing algorithms. Section 6 shows experimental results. Section 7 draws some conclusions.

2 The Cell Cycle Model

This section explains Gillespie's stochastic simulation algorithm first, and then describes implementation aspects of the budding yeast cell cycle model. The multistage cell tracking algorithm is also discussed.

2.1 The Stochastic Simulation Algorithm (SSA)

Consider a biochemical system or pathway that involves N molecular species $S_1, \ldots, S_N$. $X_i(t)$ denotes the number of molecules of species S_i at time t. Stochastic simulations generate the evolution of the state vector $X(t) = (X_1(t), ..., X_N(t))$ given that the system was initially in the state vector $X(t_0)$. Suppose the system is composed of M reaction channels $R_1, \ldots, R_M$. In a constant volume Ω, assume that the system is well-stirred and in thermal equilibrium at some constant temperature. There are two important entities in reaction channel R_j: the state change vector $\nu_{\cdot j} = (\nu_{1j}, ..., \nu_{Nj})$, and the propensity function a_j. ν_{ij} is defined as the change in the S_i molecules' population caused by one R_j reaction. $a_j(x)dt$ gives the probability that one R_j reaction will occur in the next infinitesimal time interval $[t, t + dt)$.

The SSA simulates every reaction event [7]. $X(t) = x$, $p(\tau, j|x, t)d\tau$ is defined as the probability that the next reaction in the system will occur in the infinitesimal time interval $[t+\tau, t+\tau+d\tau)$, and will be an R_j reaction. By letting $a_0(x) \equiv \sum_{j=1}^{M} a_j(x)$, the equation

$$p(\tau, j|x, t) = a_j(x) \exp(-a_0(x)\tau)$$

can be obtained. A Monte Carlo method is used to generate τ and j. On each step of the SSA, random numbers r_1 and r_2 are generated from the uniform (0,1) distribution. From probability theory, the time for the next reaction to occur is given by $t+\tau$, where

$$\tau = \frac{1}{a_0(\mathbf{x})} \ln\left(\frac{1}{r_1}\right).$$

The next reaction index j is given by the smallest integer satisfying

$$\sum_{j'=1}^{j} a_{j'}(x) > r_2 a_0(x).$$

After τ and j are obtained, the system states are updated by $X(t+\tau) := x+\nu_j$, and the time is updated by $t := t+\tau$. This simulation proceeds iteratively until the time t reaches its termination value.

It is clear from the stochastic nature of the system that a different simulation of the same cell over the same interval (using a different seed for the pseudo-random number generator) will involve a different number of reactions, and therefore will require a different compute time.

2.2 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

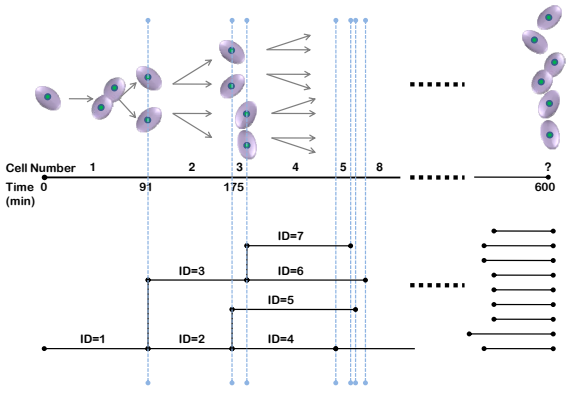


Fig. 1. Multistage cell cycle tracking diagram. ID is the cell identification tag. Cell modeling simulations should be executed beginning at each cell emergence time. Biologists are interested in how many cells exist at a specific final time.

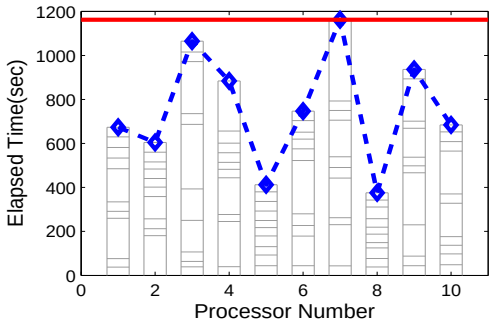


Fig. 2. Elapsed compute times for 100 prototype multistage cell cycle simulations by static distribution across 10 worker processors.

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.

Molecular biologists have dissected and characterized individual components and their interactions to derive a consensus picture of the regulatory network of budding yeast. The mechanism controls the activity of three important classes of cyclins: Cln2, Clb5, and Clb2. Cln2 is primarily responsible for bud emergence, Clb5 for initiating DNA synthesis, and Clb2 for driving the cell into mitosis. To exit mitosis, all Clb-dependent kinase activity must be destroyed, which is the job of Cdc20, Cdc14, and Sic1 [5], [8].

Stochastic methods require the model to be expressed in terms of population numbers because they consider reactions with individual molecules. Because the original budding yeast model [5] is based on normalized concentration values, it must first be converted to a model based on numbers of molecules [6]. The JigCell Model Builder (JCMB) [16] allows the user to create and modify models with a simple spreadsheet-like interface. JCMB includes a tool to convert the concentration-based cell cycle model to the number-based model. The physical model is in the Systems Biology Markup Language (SBML) format [17]. After creating the population-based budding yeast model, Gillespie's SSA is executed with the multistage cell tracking method that is explained in detail in the next section. Conversion and simulation steps are presented in [6], [18], [19].

2.3 Multistage Cell Tracking Implementation

To accurately mimic the experimental protocol it is necessary to simulate all of the progeny from a single cell that is chosen from a specific distribution of initial conditions. Existing stochastic simulators using the Gillespie SSA just simulate a system with one initial molecular state vector. To simulate all of the progeny, whose initial states are different, multicycle cell lineage tracking is needed.

Figure 1 symbolizes the multistage cell cycle implementation. Implementing the multistage cell cycle simulation requires selecting the initial conditions for subsequent simulations. A priority queue is used to maintain the cell division events ordered by time. The precise algorithm for the multistage cell cycle implementation follows. If one cell divides well into two daughter cells, then the program inserts two absolute times and indexes for these two cells into the priority queue. The daughter cells' number of molecules at division are stored in a repository matrix along with the cell index. After one simulation completes, the next simulation index chosen will be that with the smallest time in the priority queue. The next simulation takes the initial molecular state vector read from the matrix. The multistage cell tracking implementation continuously simulates every cell lineage cycle until the simulation's termination time.

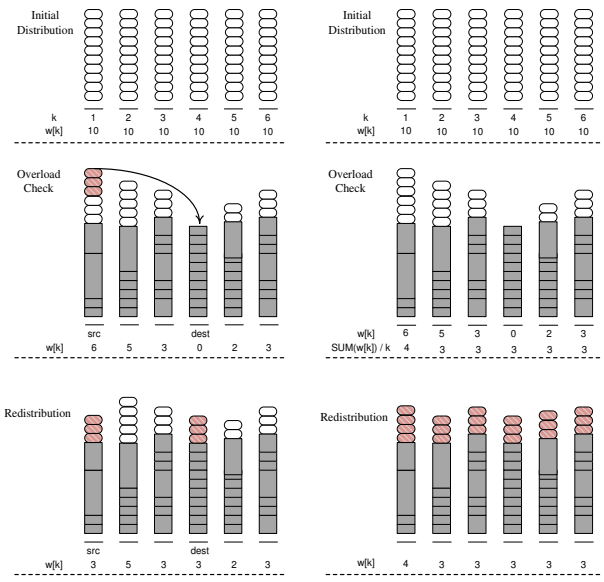
3 Load Balancing Algorithms

This section presents two dynamic load balancing strategies: the point-to-point (P2P) algorithm and the all-redistribution (AR) algorithm.

3.1 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 dramatic differences in number of cells among cell lineages can cause a severe load imbalance among processors that are each simulating many lineages.

Consider, for example, stochastic simulations of the budding yeast cell. For certain mutants, a cell may never divide, or it may divide several cell cycles partially that means cells divide with some probability of



(a) P2P load balancing idea. (b) AR load balancing idea.

Fig. 3. Adaptive load balancing strategies. Ellipses represent tasks to be done and gray rectangles represent completed tasks. Gray elipses indicate tasks to be done on processors whose load has been adjusted by an adaptive load balancing algorithm.

dividing. Therefore, the CPU time to simulate such a mutant cell is quite different from one case to another. Figure 2 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.

3.2 Point-to-Point Algorithm

The P2P algorithm is based on the *central redistribution* work of Powley [20] and Hillis [21]. The idea of the P2P algorithm is presented in Figure 3 (a). First, the tasks (cell simulations) are evenly distributed to every worker processor in 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 this P2P 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 *termination*. The termination detection approach used here is that described by Bertsekas and Tsitsiklis [11], using a request and acknowledgement message. Initially, each processor is in one of two states: inactive and active. Upon receiving a task from the master, slave processors are active. Slave processors 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, reduces its tasks to half of its remaining jobs, and then resumes execution where it left off.

The advantage of a centralized dynamic load balancing algorithm is that the master processor can easily recognize when to terminate. A slave process can detect when the local work is complete, but cannot easily detect which remote processor has completed its work, or which has the highest load. The disadvantage is that the approach does not scale, since the master becomes a bottleneck and must maintain a large amount of global state information. Centralized load balancing is an adequate approach for stochastic ensemble simulations when the jobs are long compared to the communication time.

3.3 All-Redistribution Algorithm

The all-redistribution (AR) algorithm is also a centralized load balancing scheme. The idea of the AR algorithm is presented in Figure 3 (b). The initial step of the AR algorithm is similar to that of the P2P algorithm. The processor that finished 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.

4 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 in detail.

4.1 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 $R_1 + \ldots + R_p = n$. 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 [22]. Therefore, using Assumption 1, if R_i is large enough, then

$$\frac{X_i - R_i\,\mu_T}{\sqrt{R_i}\,\sigma_T}$$

will be approximately normally distributed with

$$\mathrm{E}\left[X_i\right] = R_i \cdot \mu_T \ , \quad \mathrm{V}ar\left[X_i\right] = R_i \cdot \sigma_T^2 \ .$$

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.

4.2 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

$$\xi_X^2 = \frac{1}{p-1}\sum_{i=1}^{p}\left(X_i - \eta_X\right)^2$$

and is also a random variable. 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,

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

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

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,

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

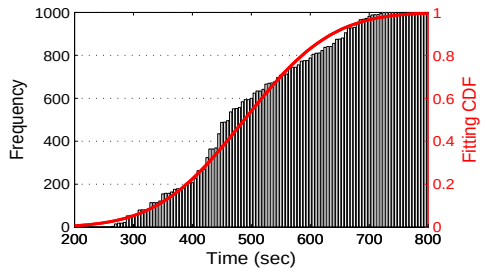
Finally, the third measure of load imbalance is the expected value of the average idle compute time across all processors,

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

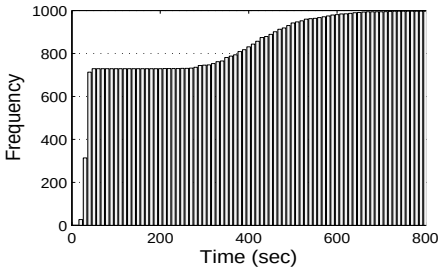
4.3 Variability in Compute Times per Cell

The wild-type cell lineage simulation time distribution from a simulation experiment is plotted in Figure 4 (a). This distribution is based on 1,000 budding yeast multistage cell tracking simulations with 25 processors. 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 [23]. The wild-type simulation data from Figure 4 (a) has the mean and standard deviation

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



(a) Wild-type 1,000 simulations.



(b) Prototype mutant 1,000 simulations.

Fig. 4. Discrete cumulative histogram of compute times per cell (grey bar) for wild-type and mutant simulations. The solid line represents the best-fit Gaussian CDF.

Figure 4 (b) 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

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

5 Analysis of the Load Balancing Algorithms

The probabilistic framework proposed here models compute times per task as i.i.d. random variables. 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.

This analysis framework is useful for a considerably more general class of problems, beyond stochastic cell cycle modeling. The proposed analysis approach is applicable to any parallel ensemble calculations where the compute times per task follow the same probability distribution.

5.1 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 [24] follow. The CDF of the largest order statistic Y_p is given by

$$\begin{aligned} F_{Y_p}(y) &= \Pr[\, Y_p \leq y \,] = \Pr[\, X_1 \leq y; \ldots; X_p \leq y \,] \\ &= \prod_{j=1}^{p} \Pr[\, X_j \leq y \,] = \prod_{j=1}^{p} F_{X_j}(y) = [\, F_X(y) \,]^p \end{aligned}$$

because the X_js are independent. Likewise

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

These are important special cases of the general formula for $F_{Y_r}(y)$,

$$\begin{aligned} F_{Y_r}(y) &= \Pr[\, Y_r \leq y \,] \\ &= \sum_{i=r}^{p} \binom{p}{i} [F_X(y)]^i \, [1 - F_X(y)]^{p-i}. \end{aligned}$$

The probability density function for the rth order variable Y_r from X is

$$f_{Y_r}(y) = \frac{[F_X(y)]^{r-1}[1 - F_X(y)]^{p-r}}{\mathrm{B}(r, p-r+1)} f_X(y),$$

where $\mathrm{B}(a,b) = \dfrac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$ is the Euler beta function. $\Gamma(r) = \displaystyle\int_0^\infty x^{r-1} e^{-x}\, dx$ is the gamma function. 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 [25] 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 $\mathrm{erf}(x) = \dfrac{2}{\sqrt{\pi}} \displaystyle\int_0^x e^{-t^2}\, dt$. 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 [25] 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 Section 6 indicate that the approximations (6a) and (6b) are very close to the experimentally determined expected values.

5.2 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). 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

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

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

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

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

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

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

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

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*

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

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;

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

Now

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

Recall that $\widehat{R}_i$ 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

$$\mathrm{E}\left[\sum_{j=1}^{R_i} (t_{ij} - \mathrm{E}[t_{ij}])\right] = 0\ .$$

$\mathrm{E}[(X_i - \eta_X)^2]$ will be determined from (10). First apply Lemma 2 to get

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

In compact notation

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

where δ_{i1} is the Kronecker delta. Due to the independence of individual compute times,

$$\mathrm{E}\left[(t_{ij} - \mathrm{E}[t_{ij}])\,(t_{\ell m} - \mathrm{E}[t_{\ell m}])\right] = 0 \text{ for } j \neq m \text{ or } i \neq \ell.$$

Hence $\mathrm{E}[(X_i - \eta_X)^2] =$

$$\left(\widehat{R}_i - \mathrm{M}(\widehat{R})\right)^2 \mu_T^2 + \frac{1}{p} \left(\mathrm{M}(\widehat{R}) + (p-2)\, \widehat{R}_i\right) \sigma_T^2 + \left(-\frac{1}{6}\, \sigma_T^2 + \frac{1}{12}\, \mu_T^2\right) \left(\frac{p^2 - p - 1 - (p^2 - 2p)\, \delta_{i1}}{p^2}\right).$$

Finally the expected value of the algebraic variance

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

□

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).

5.3 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 $\widehat{R}_i = R$,

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

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

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

$$\mathrm{E}[Y_1] = \int_{-\infty}^{\infty} y\; p\,[1-F_X(y)]^{p-1} f_X(y)\; dy \quad (15)$$

with the Gaussian probability density function

$$f_X(y) = \frac{1}{\sigma_X\sqrt{2\pi}} e^{-(y-\mu_X)^2/(2\sigma_X^2)}\,, \quad (16)$$

and the Gaussian cumulative distribution function

$$F_X(y) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{y-\mu_X}{\sigma_X\sqrt{2}}\right)\right] . \quad (17)$$

From (14)–(17) 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

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

and

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

5.4 Analysis of P2P 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 P2P 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 P2P 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 P2P 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

$$\mathrm{E}[X_1'] = R_1'\,\mu_T = \left\lfloor \frac{R_p}{2} \right\rfloor\,\mu_T\,,$$

$$\mathrm{E}[X_p'] = \left(R_p' - \frac{1}{2}\right)\mu_T = \left(\left\lceil \frac{R_p}{2} \right\rceil - \frac{1}{2}\right)\mu_T\,.$$

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 P2P 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 P2P load balancing step by*

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

Proof: The average adjusted number of tasks per processor is the same before and after P2P load balancing, $\mathrm{M}(\widehat{R}) = \mathrm{M}(\widehat{R}')$. The decrease in the algebraic variance of the adjusted number of tasks is

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

Lemma 4 provides the difference between the expected variances of compute times across processors before and after a P2P load balancing step,

$$\mathrm{E}[\xi_X^2] - \mathrm{E}[\xi_{X'}^2] = \frac{R_p\,(R_p-1)}{2\,(p-1)}\,\mu_T^2\ . \quad (18)$$

The P2P algorithm can be meaningfully applied only when the number of tasks on the most overloaded processor is $R_p \ge 2$. The relation (18) 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 P2P load balancing step,* $\mathrm{E}\left[Y_p' - Y_1'\right] \le \mathrm{E}\left[Y_p - Y_1\right]$.

Proof: Before the P2P load balancing step the expected maximum imbalance is

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

After the P2P load balancing step, the new expected maximum imbalance time is $\mathrm{E}[Y_p'] - \mathrm{E}[Y_1']$.

Consider the random variables

$$Z_2 = \min\{X_2, \cdots, X_{p-1}\}\,,$$

$$Z_{p-1} = \max\{X_2, \cdots, X_{p-1}\} \le Y_p\,.$$

The smallest and the largest order statistics after P2P balancing are $Y'_1 = \min\{X'_1, X'_p, Z_2\}$ and $Y'_p = \max\{X'_1, X'_p, Z_{p-1}\}$. 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 load balancing. The remaining seven combinations are as follows:

(1) $Y'_1 = Z_2$ and $Y'_p = Z_{p-1}$;
(2) $Y'_1 = Z_2$ and $Y'_p = X'_p$;
(3) $Y'_1 = Z_2$ and $Y'_p = X'_1$;
(4) $Y'_1 = X'_p$ and $Y'_p = Z_{p-1}$;
(5) $Y'_1 = X'_p$ and $Y'_p = X'_1$;
(6) $Y'_1 = X'_1$ and $Y'_p = Z_{p-1}$;
(7) $Y'_1 = X'_1$ and $Y'_p = X'_p$.

In Case (1) the balanced times fall between Z'_2 and Z'_{p-1}. The expected maximum idle time reduction is

$$\begin{aligned}&\{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\}\\ &\quad = \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Z_{p-1}] - \mathrm{E}[Z_2]\}\\ &\quad = \{\mathrm{E}[Y_p] - \mathrm{E}[Z_{p-1}]\} + \{\mathrm{E}[Z_2]\} \;\geq\; \mathrm{E}[Z_2] \;\geq\; 0\,.\end{aligned}$$

The reductions of expected maximum idle times for Cases (2) to (7) are straightfoward verification.

$$\begin{aligned}&\mathrm{Case(2):}\;\; \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\}\\ &\quad = \mathrm{E}[Y_p] - \{\mathrm{E}[X'_p] - \mathrm{E}[Z_2]\} \;\geq\; \mathrm{E}[Y_p] - \mathrm{E}[X'_p]\\ &\quad \geq \widehat{R}_p\,\mu_T - (\lceil R_p/2\rceil - 0.5)\,\mu_T > 0\,.\\ &\mathrm{Case(3):}\;\; \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\}\\ &\quad = \mathrm{E}[Y_p] - \{\mathrm{E}[X'_1] - \mathrm{E}[Z_2]\} \;\geq\; \mathrm{E}[Y_p] - \mathrm{E}[X'_1]\\ &\quad \geq (R_p - 0.5 - \lfloor R_p/2\rfloor)\;\mu_T \;>\; 0\,.\\ &\mathrm{Case(4):}\;\; \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\}\\ &\quad = \mathrm{E}[Y_p] - \{\mathrm{E}[Z_{p-1}] - \mathrm{E}[X'_p]\} \;\geq\; \mathrm{E}[X'_p]\\ &\quad = (\lceil R_p/2\rceil - 0.5)\;\mu_T \;>\; 0\,.\\ &\mathrm{Case(5):}\;\; \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\}\\ &\quad = \mathrm{E}[Y_p] - \{\mathrm{E}[X'_1] - \mathrm{E}[X'_p]\}\\ &\quad \geq \left(\widehat{R}_p - \lfloor R_p/2\rfloor + \lceil R_p/2\rceil - 0.5\right)\;\mu_T \;>\; 0\,.\\ &\mathrm{Case(6):}\;\; \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\}\\ &\quad = \mathrm{E}[Y_p] - \{\mathrm{E}[Z_{p-1}] - \mathrm{E}[X'_1]\} \;\geq\; \mathrm{E}[X'_1] \;>\; 0\,.\\ &\mathrm{Case(7):}\;\; \{\mathrm{E}[Y_p] - \mathrm{E}[Y_1]\} - \{\mathrm{E}[Y'_p] - \mathrm{E}[Y'_1]\}\\ &\quad = \mathrm{E}[Y_p] - \{\mathrm{E}[X'_p] - \mathrm{E}[X'_1]\}\\ &\quad \geq \left(\widehat{R}_p + 0.5 + \lfloor R_p/2\rfloor - \lceil R_p/2\rceil\right)\mu_T \;>\; 0\,.\end{aligned}$$

Therefore, after a P2P load balancing step, the expected maximum time imbalance is always the same on reduced. □

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

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

PROPOSITION 3. *The expected value of the average idle time (3) does not increase after a P2P load balancing step,* $\mathrm{E}\left[Y'_p - \eta_{X'}\right] \leq \mathrm{E}\left[Y_p - \eta_X\right]$.

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.

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

□

5.5 Analysis of AR 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, $i = 2, \cdots, p$, 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 $b = \left(\sum_{i=1}^{p} R_i\right) \mod p$, and

$$r = \lfloor \mathrm{M}(R)\rfloor = \mathrm{M}(R) - \frac{b}{p}, \quad \widehat{r} = r - \frac{1}{2}\,.$$

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

$$\mathrm{E}[X'_i] = \begin{cases} r\,\mu_T\,, & \text{if } i = 1\,,\\ \widehat{r}\,\mu_T\,, & \text{if } i = 2, \cdots, p-b\;,\\ (\widehat{r}+1)\,\mu_T\,, & \text{if } i = p-b+1, \ldots, p\,.\end{cases} \tag{19}$$

PROPOSITION 4. *The expected algebraic variance (1) of X' is smaller than the expected algebraic variance of X after an AR load balancing step,* $\mathrm{E}[\xi^2_{X'}] < \mathrm{E}[\xi^2_X]$.

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 of the modified number of jobs after AR load balancing is

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

The decrease in the expected value of the algebraic variance of the compute times is

$$\mathrm{E}\left[\xi_X^2\right] - \mathrm{E}\left[\xi_{X'}^2\right] = \left(\mathrm{V}(\widehat{R}) - \frac{p\,(4b+1)-(2b+1)^2}{4\,p\,(p-1)}\right)\mu_T^2\,.$$

□

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 load balancing 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 load balancing step,* $\mathrm{E}\left[Y'_p - Y'_1\right] < \mathrm{E}\left[Y_p - Y_1\,.\right]$

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

$$\mathrm{E}[Y_p] \geq \mathrm{E}[X_p] = R_p\,\mu_T\ .$$

Similarly, the minimum compute time is at most equal to the compute time on the processor with the smallest number of remaining jobs. Therefore

$$\mathrm{E}[Y_1] \leq \mathrm{E}[X_1] = R_1\,\mu_T = 0\ ,$$

and

$$R_p\,\mu_T \leq \mathrm{E}[Y_p - Y_1]\ , \quad \big(R_p - M(R)\big)\,\mu_T \leq \mathrm{E}[Y_p - \eta_Y]\ .$$

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 (19),

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

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

$$\begin{aligned} |\mathrm{err_p}(p)| &\leq \epsilon \cdot \left| r\,\mu_T + \sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p})\right|\ , \\ |\mathrm{err}_1(p)| &\leq \epsilon \cdot \left| r\,\mu_T - \sqrt{r}\,\sigma_T\,\Phi^{-1}(0.5264^{1/p})\right|\ , \end{aligned}$$

taking the relative errors with respect to the approximate values for convenience. Note that the results in [25] 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,

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

The decrease in expected maximum idle time is at least

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

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

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 $\mathrm{E}\left[Y_p - \eta_X\right] \geq (R_p - r)\,\mu_T$. 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

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

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

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

□

6 Experimental Results

This section provides experimental load balancing results with the budding yeast cell cycle model. To evaluate the proposed load balancing algorithms, the ensemble of simulations is executed on Virginia Tech's System X supercomputer [26]. The supercomputer has 1,100 Apple PowerMac G5 nodes, with dual 2.3 GHz PowerPC 970FX processors and 4GB memory.

6.1 Evaluation of the Static Balancing

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 asses how well the theoretical estimates of load imbalance metrics agree with the simulation results. 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 (14)–(15) and the approximation method (6a)–(6b). $\mathrm{E}[Y_p] = 20,973$ and $\mathrm{E}[Y_1] = 18,075$ from the former method are very similar to the latter approximation results $\mathrm{E}[Y_p] = 20,965$ 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,

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

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

6.2 Evaluation of Load Balancing Efficiency

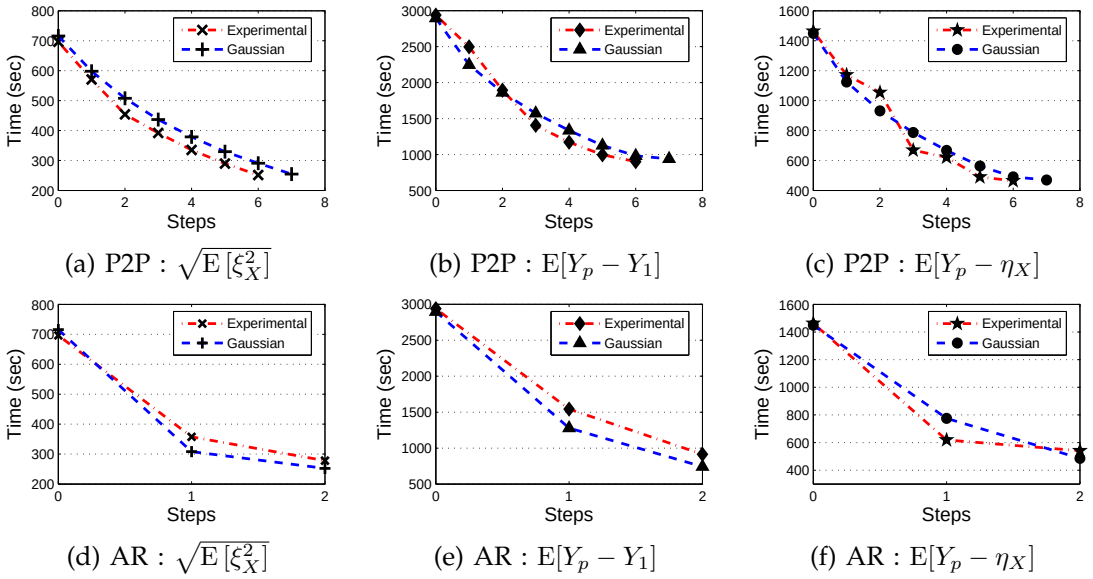
Two test cases are used to numerically evaluate the P2P and the AR algorithms: the first uses the real compute times per task obtained from running the wild-type cell cycle model, and the second uses synthetic task compute times drawn from the best fit Gaussian distribution. There are $n = 1,000$ tasks in the ensemble run on $p = 25$ processors.

Figures 5 (a) and (d) show the decrease in the root expected algebraic variance of compute times when performing several P2P and AR steps of the algorithm. Figures 5 (b) and (e) show the reduction in the expected maximum load imbalance among CPUs. Figures 5 (c) and (f) show the decrease in the expected idle time per processor after load balancing steps. For all test cases the variance decreases consistently on each iteration as predicted by the theory. The AR algorithm requires fewer iterations than the P2P algorithm.

6.3 Load Balancing Results for Wild-Type Yeast

This section describes our load balancing results for the wild-type budding yeast cell cycle model. Figure 6 compares the processor CPU times and wall-clock time using no dynamic load balancing, P2P load balancing, and AR load balancing algorithms. 1,000 simulations with 25 processors and 10,000 simulations with 100 processors are executed for this experiment. With the static distribution (no load balancing), the variance of the CPU times is not huge because wild-type cells divide in a relatively regular fashion. The simulation time is just affected by the stochastic nature of the SSA. Nevertheless, the two dynamic load balancing methods reduce the wall-clock time compared to the static method; the differences are approximately 1,000 seconds (4.9% of static distribution wall-clock time) for 1,000 runs with 25 processors, and 2,800 seconds (5.4% of static distribution wall-clock time) for 10,000 runs with 100 processors.

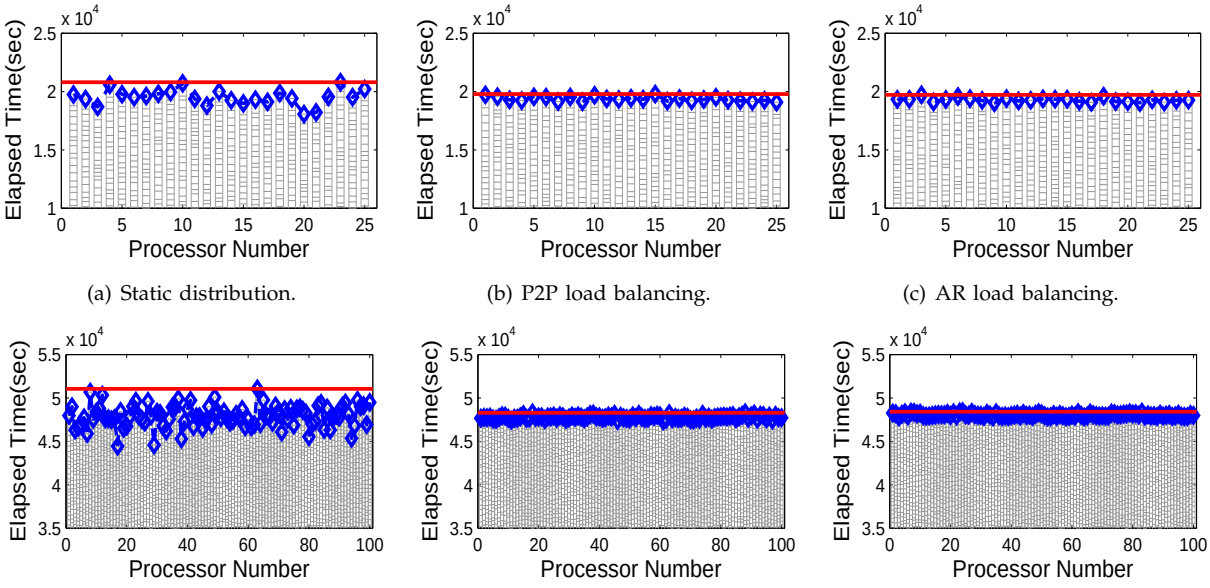
Table 1 demonstrates the efficiency of the two dynamic load balancing algorithms clearly. The average idle CPU times (3) for the no-balancing simulation are 1270.75 seconds for 1,000 runs with 25 processors and 3160.77 seconds for 10,000 runs with 100 processors. Therefore, the average idle CPU time has increased with the increasing number of jobs per processor. The average idle CPU times for the load balancing algorithms, however, do not increase with the number of jobs per processor. For the P2P load balancing simulation, the average idle times are 418.88 seconds for 1,000 runs with 25 processors and 510.84 seconds for 10,000 runs with 100 processors. For the AR load balancing simulation, the average idle times are 432.44 seconds for 1,000 runs with 25 processors and 373.80 seconds for 10,000 runs with 100 processors. Figure 7 compares the average idle CPU times for the static and load balancing algorithms. For wild-type yeast



(a) P2P : $\sqrt{\mathrm{E}\left[\xi_X^2\right]}$ (b) P2P : $\mathrm{E}[Y_p - Y_1]$ (c) P2P : $\mathrm{E}[Y_p - \eta_X]$

(d) AR : $\sqrt{\mathrm{E}\left[\xi_X^2\right]}$ (e) AR : $\mathrm{E}[Y_p - Y_1]$ (f) AR : $\mathrm{E}[Y_p - \eta_X]$

Fig. 5. Numerical evaluation for the two load-balancing algorithms on two test cases: with the real simulation times per task and with synthetic Gaussian distributed simulation times per task. There are $n = 1,000$ tasks in the ensemble run on $p = 25$ processors. The root expected algebraic variance, the expected maximum imbalance, and the expected idle CPU time per processor are evaluated with the P2P and AR load balancing algorithms.



(a) Static distribution. (b) P2P load balancing. (c) AR load balancing.

(d) Static distribution. (e) P2P load balancing. (f) AR load balancing.

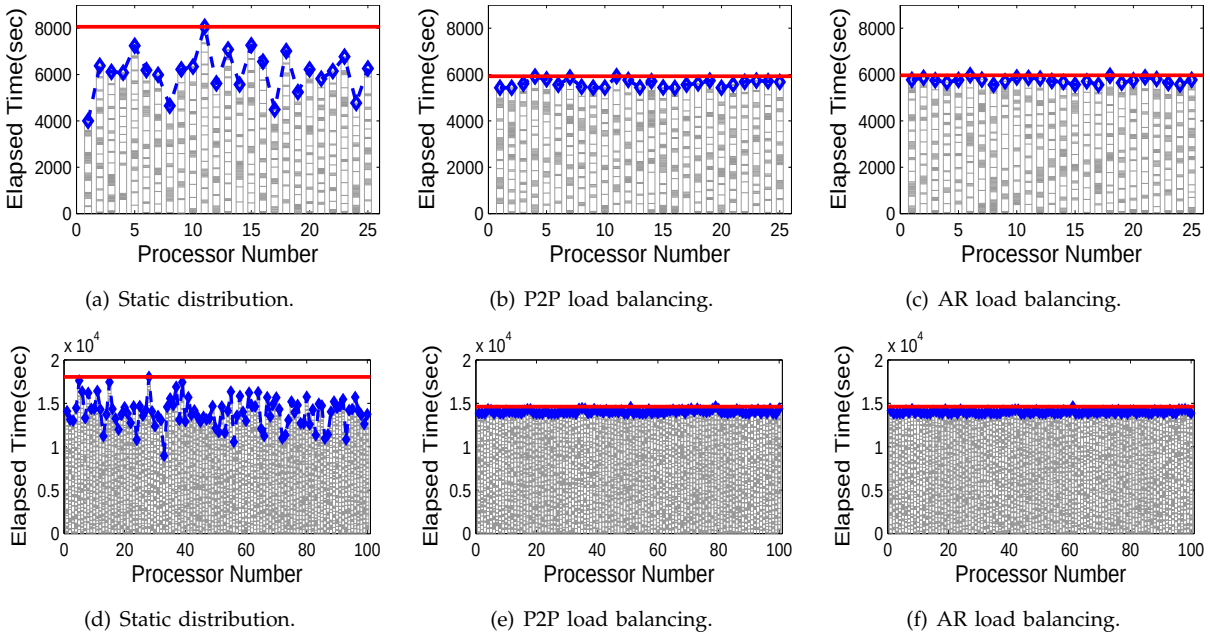
Fig. 6. Elapsed compute times per processor (diamond marker) and wall clock time (solid line) for wild-type multistage cell cycle simulations. 1,000 runs with 25 processors for (a), (b), (c) and 10,000 runs with 100 processors for (d), (e), (f). Small grey rectangular height represents each job time for the processor.

simulations, the dynamic load balancing 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 two dynamic load balancing algorithms are approximately 0.2 seconds for 1,000 runs with 25 processors and 1.0 second for 10,000 runs with 100 processors. Therefore, the total communication time for the load balancing is negligible compared to elapsed wall-clock time. The two load balancing algorithms (P2P and AR) have similar performance for these simulations. Both of the load balancing strategies reduce system resources efficiently for the wild-type cell cycle simulation.

TABLE 1
Average, maximum, and minimum compute times, maximum idle time, average (percentage) idle time and RAV (root of the algebraic variance) of compute times for wild-type cell simulations. The static and the two proposed load balancing approaches are compared by results from both a small and a large ensemble. Units are seconds.

	1,000 Runs (25 processors)			10,000 Runs (100 processors)		
Metrics	Static	P2P	AR	Static	P2P	AR
Avg compute time	19524.46	19362.33	19276.73	47880.41	47778.01	48038.93
RAV of compute times	679.83	195.10	171.67	1271.90	196.16	155.76
Max compute time	20795.21	19781.22	19708.78	51050.18	48288.85	48412.73
Min compute time	18054.79	19084.42	19033.22	44430.84	47354.04	47801.99
Max idle time	2740.42	696.80	675.56	6619.34	934.81	610.74
Avg idle time	1270.75	418.89	432.05	3169.77	510.84	373.80
Percentage idle time (%)	6.50%	2.16%	2.24%	6.62%	1.07%	0.78%



(a) Static distribution. (b) P2P load balancing. (c) AR load balancing.

(d) Static distribution. (e) P2P load balancing. (f) AR load balancing.

Fig. 8. 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), (b), (c) and 10,000 runs with 100 processors for (d), (e), (f). Small grey rectangular height represents each job time for the processor.

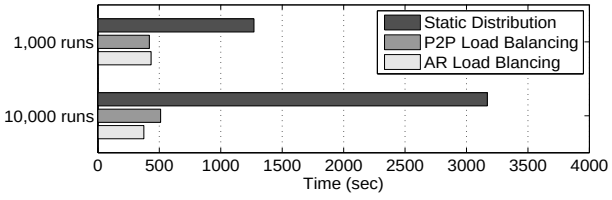


Fig. 7. The average idle CPU times comparison for the static distribution and the final step of the load balancing methods.

6.4 Load Balancing Results for Mutant Yeast

This section presents experimental results for the prototype budding yeast mutant cell cycle model. For the mutant strain we are considering, the initial cell might never divide at all or it might divide several times and then cease division [5]. 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 two dynamic load balancing algorithms show huge advantages in CPU utilization.

Figure 8 compares the CPU time that each processor required to complete its assigned tasks. Figures 8 (a), (b), (c) show results for 1,000 runs with 25 processors and Figures 8 (e), (f), (g) show results for 10,000 runs with 100 processors. For the static distribution, the variance is huge because of the characteristics of mutant simulations. Different rectangle heights show different CPU times per task. The dynamic load balancing algorithms reduced 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. Thus the two dynamic load balancing

TABLE 2
Average, maximum, and minimum compute times, maximum idle time, average (percentage) idle time and RAV (root of the algebraic variance) of compute times for wild-type cell simulations. The static and the two proposed load balancing approaches are compared by results from both a small and a large ensemble. Units are seconds.

	1,000 Runs (25 processors)			**10,000 Runs (100 processors)**		
Metrics	Static	P2P	AR	Static	P2P	AR
Avg compute time	6079.19	5611.64	5740.47	13920.60	14041.65	13989.86
RAV of compute times	943.35	165.24	117.51	1695.19	192.80	164.07
Max compute time	8054.18	5922.13	5965.04	18037.66	14616.97	14605.74
Min compute time	3995.03	5416.86	5553.87	8949.82	13801.48	13814.88
Max idle compute time	4059.15	505.27	411.17	9087.84	815.49	790.86
Avg idle time	1974.99	310.49	224.58	4117.06	575.32	615.88
Percentage idle time (%)	32.49%	5.53%	3.91%	29.57%	4.10%	4.40%

algorithms show greater improvements for mutant model simulation than for wild-type model simulation. Table 2 also shows the improved efficiency of the two dynamic load balancing 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).

7 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 that the compute times of individual tasks are not known, but 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 underlying probability density; therefore the analysis is very general. 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 two dynamic load balancing strategies. In the P2P algorithm the idle processor receives new tasks from the most overloaded processor. In the AR algorithm the master processor redistributes all jobs evenly to the workers when one processor finishes all its work. Termination detection is using request and acknowledgement messages. The analysis reveals that the expected level of load imbalance is decreased after a 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 reduced the total compute times by about 5% for ensembles of wild type cells, and by about 25% for ensembles of mutant cells. Average processor idle time was reduced by 85% or more for ensembles of mutant cells.

Future work will investigate decentralized dynamic load balancing approaches that are scalable to massively parallel architectures. The theoretical analysis proposed here for centralized schemes will be extended to the decentralized case. Scalability, not investigated here, will also be analyzed. It would be valuable to simulate and analyze large ensemble runs with different models, where the i.i.d. assumption does not hold.

Acknowledgments

This work is supported by awards NIGMS/NIH 5 R01 GM078989, AFOSR FA9550–09–1–0153, NSF DMS–0540675, NSF CCF–0916493, and NSF OCI–0904397.

References

[1] H. H. McAdams and A. Arkin, "Stochastic mechanisms in gene expression," *Proc. Natl. Acad. Sci.*, vol. 94, pp. 814–819, 1997.

[2] J. M. Murphy, D. M. H. Sexton, D. N. Barnett, G. S. Jones, M. J. Webb, M. Collins, and D. A. Stainforth, "Quantification of modelling uncertainties in a large ensemble of climate change simulations," *Nature*, vol. 430, pp. 768–772, 2004.

[3] V. Nefedova, R. Jacob, I. Foster, Z. Liu, Y. Liu, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Automating climate science: large ensemble simulations on the teragrid with the GriPhyN virtual data system," in *Proc. of the Second IEEE Int. Conf. on e-Science and Grid Computing (E-SCIENCE '06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 32–37.

[4] A. Murray and T. Hunt, *The Cell Cycle: an Introduction*. New York, USA: Oxford University Press, 1993.

[5] K. C. Chen, L. Calzone, A. Csikasz-Nagy, F. R. Cross, B. Novak, and J. J. Tyson, "Integrative analysis of cell cycle control in budding yeast," *Mol. Biol. Cell*, vol. 15, no. 8, pp. 3841–3862, 2004.

[6] P. Wang, R. Randhawa, C. A. Shaffer, Y. Cao, and W. T. Baumann, "Converting macromolecular regulatory models from deterministic to stochastic formulation," in *Proceedings of the 2008 Spring simulation multiconference*. San Diego, CA, USA: Society for Computer Simulation International, 2008, pp. 385–392.

[7] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *J. Phys. Chem.*, vol. 81, no. 25, pp. 2340–2361, 1977.

[8] K. C. Chen, A. Csikasz-Nagy, B. Gyorffy, J. Val, B. Novak, and J. J. Tyson, "Kinetic analysis of a molecular model of the budding yeast cell cycle," *Mol. Biol. Cell*, vol. 11, no. 1, pp. 369–391, 2000.

[9] W. W. Chu, L. J. Holloway, M.-T. Lan, and K. Efe, "Task allocation in distributed data processing," *Computer*, vol. 13, no. 11, pp. 57–69, 1980.

[10] M. A. Iqbal, J. H. Saltz, and S. H. Bokhari, "A comparative analysis of static and dynamic load balancing strategies," *ACM Performance Evaluation Revision*, vol. 11, no. 1, pp. 1040–1047, 1985.

[11] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[12] B. P. Lester, *The Art of Parallel Programming*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[13] J. Jacob and S.-Y. Lee, "Task spreading and shrinking on a network of workstations with various edge classes," in *Proc. 1996 Int'l Conf. Parallel Processing*, vol. 3, Aug. 1996, pp. 174–181 vol.3.

[14] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2002.

[15] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[16] M. T. Vass, C. A. Shaffer, N. Ramakrishnan, L. T. Watson, and J. J. Tyson, "The JigCell model builder: A spreadsheet interface for creating biochemical reaction network models," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 3, no. 2, pp. 155–164, 2006.

[17] M. Hucka, A. Finney, H. Sauro, and 40 additional authors, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinfomatics*, vol. 19, no. 4, pp. 524–531, 2003.

[18] T.-H. Ahn, P. Wang, L. T. Watson, Y. Cao, C. A. Shaffer, and W. T. Baumann, "Stochastic cell cycle modeling for budding yeast," in *Proceedings of the 2009 Spring Simulation Multiconference*, ser. SpringSim '09. San Diego, CA, USA: Society for Computer Simulation International, 2009, pp. 113:1–113:6.

[19] T.-H. Ahn, L. T. Watson, Y. Cao, C. A. Shaffer, and W. T. Baumann, "Cell cycle modeling for budding yeast with stochastic simulation algorithms," *Computer Modeling in Engineering and Sciences*, vol. 51, no. 1, pp. 27–52, 2009.

[20] C. Powley, C. Ferguson, and R. E. Korf, "Depth-first heuristic search on a simd machine," *Artif. Intell.*, vol. 60, no. 2, pp. 199–242, 1993.

[21] W. D. Hillis, *The Connection Machine*. Cambridge, MA, USA: MIT Press, 1986.

[22] J. A. Rice, *Mathematical Statistics and Data Analysis*, 3rd ed. Belmont, CA, USA: Duxbury Press, 2001.

[23] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, 2nd ed. Hoboken, NJ, USA: Wiley-Interscience, 2001.

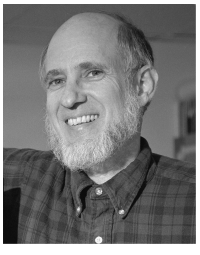
[24] H. A. David and H. N. Nagaraja, *Order Statistics*, 2nd ed. Hoboken, NJ, USA: Wiley-Interscience, 2003.

[25] C.-C. Chen and C. W. Tyler, "Accurate approximation to the extreme order statistics of gaussian samples," *Communications in Statistics - Simulation and Computation*, vol. 28, no. 1, pp. 177–188, 1999.

[26] "System X Website," http://www.arc.vt.edu/arc/SystemX/.

Tae-Hyuk Ahn is a Ph.D. student in the Department of Computer Science at Virginia Tech. After he received a B.S. in Electrical Engineering from Yonsei University in S. Korea, he worked for Samsung SDS for 4 years. He received his M.S. in Electrical and Computer Engineering from Northwestern University. His current research interests are computational biology, numerical analysis, and parallel computing.

Adrian Sandu obtained the Diploma in Electrical Engineering – Control Systems from the Technical University Bucharest, Romania, M.S. in Computer Science and Ph.D. in Applied Mathematical and Computational Sciences from the University of Iowa. Between 1998–2003 he served as a faculty in the Department of Computer Science at Michigan Tech. In 2003 he joined Virginia Tech's Department of Computer Science. Sandu's research interests are in the area of computational science and engineering.



Layne T. Watson received the B.A. degree (magna cum laude) in psychology and mathematics from the University of Evansville, Indiana, in 1969, and the Ph.D. degree in mathematics from the University of Michigan, Ann Arbor, in 1974. He is a professor of computer science and mathematics at Virginia Tech. His research interests include fluid dynamics, solid mechanics, numerical analysis, optimization, parallel computation, mathematical software, image processing, and bioinformatics. He is a fellow of the IEEE, the National Institute of Aerospace, and the International Society of Intelligent Biological Medicine.

Clifford A. Shaffer received the PhD degree from the University of Maryland. He is a professor in the Department of Computer Science at Virginia Tech. His current research interests include problem-solving environments, bioinformatics, component architectures, visualization, algorithm design and analysis, and data structures. He is a senior member of the IEEE.

Yang Cao received his Ph.D. degree in computer science from the University of California, Santa Barbara in 2003. He is an Assistant Professor in the Computer Science Department at Virginia Tech. His research focuses on the development of multiscale, multiphysics stochastic modeling and simulation methods and tools that help biologists build, simulate and analyze complex biological systems. He has published around 40 refereed journal articles.

William T. Baumann received the B.S, M.S, and Ph.D. degrees in electrical engineering from Lehigh University, the Massachusetts Institute of Technology, and the Johns Hopkins University, respectively. He joined the Bradley Department of Electrical and Computer Engineering at Virginia Tech in 1985 where he is currently an associate professor. His research interests include active structural acoustic control, control of thermoacoustic instabilities, and, most recently, systems biology.