Cluster Computing 7, 177–190, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

# Genetic Algorithm Based Scheduling of Meta-Tasks with Stochastic Execution Times in Heterogeneous Computing Systems ★

ATAKAN DOĞAN
*Anadolu University, Department of Electrical and Electronics Engineering, 26470 Eskişehir, Turkey*

FÜSUN ÖZGÜNER *
*The Ohio State University, Department of Electrical Engineering, 2015 Neil Avenue, Columbus, OH 43210-1272, USA*

**Abstract.** In this study, we address the meta-task scheduling problem in heterogeneous computing (HC) systems, which is to find a task assignment that minimizes the schedule length of a meta-task composed of several independent tasks with no data dependencies. The fact that the meta-task scheduling problem in HC systems is NP-hard has motivated the development of many heuristic scheduling algorithms. These heuristic algorithms, however, neglect the stochastic nature of task execution times in an attempt to minimize a deterministic objective function, which is the maximum of the expected values of machine loads. Contrary to existing heuristics, we account for this stochastic nature by modeling task execution times as random variables. We, then, formulate a stochastic scheduling problem where the objective is to minimize the expected value of the maximum of machine loads. We prove that this new objective is underestimated by the deterministic objective function and that an optimal task assignment obtained with respect to the deterministic objective function could be inefficient in a real computing platform. In order to solve the stochastic scheduling problem posed, we develop a genetic algorithm based scheduling heuristic. Our extensive simulation studies show that the proposed genetic algorithm can produce better task assignments as compared to existing heuristics. Specifically, we observe a performance improvement on the relative cost heuristic (M.-Y. Wu and W. Shu, A high-performance mapping algorithm for heterogeneous computing systems, in: *Int. Parallel and Distributed Processing Symposium*, San Francisco, CA, April 2001) by up to 61%.

**Keywords:** meta-task, heterogeneous computing, task allocation, genetic algorithm, stochastic scheduling

## 1. Introduction

*Heterogeneous computing* (HC) systems are one of the emerging platforms to execute computationally intensive applications with diverse computing needs. One of the major challenges for harnessing the computing power of HC systems so as to achieve high performance for applications is the *scheduling problem*. In this study, we consider the meta-task scheduling problem, which is to find a task assignment that minimizes the schedule length of a meta-task (a set of independent tasks with no data dependencies). The solution of the meta-task scheduling problem, however, is not trivial and the problem is NP-hard. Thus, many heuristic algorithms including the fast greedy, min-min, and genetic algorithm [1], the segmented min-min [2], and the relative cost [3] are developed.

The heuristic algorithms proposed for scheduling meta-tasks in [1–3] rely on the expected values of execution times of tasks. However, the actual execution times of tasks rarely coincide with the expected ones in a real computing environment. For a task executing on a space-shared machine, where the task has exclusive use of the machine, the difference between the actual and expected execution times can be attributed to two factors: (1) All of the execution characteristics of the task are not known or enumerated by its developer(s). (2) The time to access memory and disk is not deterministic [4]. For a task executing on a time-shared machine, on the other hand, additional non-determinism will be present due to other tasks running on the same machine [5]. As a result, if differences between actual and expected execution times of tasks are significant, a scheduling algorithm such as those in [1–3] will be misled while making scheduling decisions, which may result in poor performance in terms of minimizing the schedule length of meta-tasks and errors in the predicted performance of meta-tasks. A few studies [4,6,7] have previously addressed this problem.

To illustrate the impact of uncertainty in the execution times of tasks on the performance of meta-task scheduling heuristics, suppose that a meta-task with two tasks, $v_1$ and $v_2$, is to be scheduled onto a heterogeneous cluster of two machines, $m_1$ and $m_2$. In addition, table 1 shows the expected and actual execution times of tasks on machines, where the actual execution times are given in parentheses. The min-min heuristic [1], for example, will assign $v_1$ to $m_1$ and $v_2$ to $m_2$ based on the expected execution times, which results in a

★ A preliminary version of this paper was published in *The 2001 International Conference on Parallel Processing Workshop on Scheduling and Resource Management for Cluster Computing*.

* Corresponding author.
E-mail: ozguner@ee.eng.ohio-state.edu

Table 1
Expected and actual execution times of tasks $v_1$ and $v_2$ on a cluster of two machines $m_1$ and $m_2$ (the actual execution times are given in parentheses).

| | $m_1$ | $m_2$ |
|---|---|---|
| $v_1$ | 5 (10) | 10 (5) |
| $v_2$ | 20 (10) | 15 (20) |

schedule length of 15 time units. This task assignment is indeed the optimal one. However, if the actual execution times differ from the expected ones as shown in table 1, the previous task assignment is not optimal any more. Assigning $v_1$ to $m_1$ and $v_2$ to $m_2$ will yield a schedule length of 20 time units, while a shorter schedule length (10 time units) could have been obtained if $v_1$ and $v_2$ had been assigned to $m_2$ and $m_1$, respectively.

Motivated by these facts, our goal in this study is to come up with an algorithm that can produce better task assignments in terms of minimizing the schedule length under the condition that the actual task execution times are not equal to the expected ones. As a first step, we model execution times of tasks by random variables so as to take the variations in the execution times of tasks into account. Based on the first step, we formulate a stochastic scheduling problem where the objective is to minimize the expected value of the maximum of machine loads. Then, we prove that this objective is underestimated by all scheduling algorithms [1–3] which attempt to minimize the maximum of the expected values of machine loads. We also prove that even an optimal task assignment obtained by ignoring the variances in the execution times of tasks could be far from the actual optimal solution. In order to solve the stochastic scheduling problem posed, we develop a genetic algorithm based scheduling heuristic and show by extensive simulation studies that the proposed genetic algorithm can produce better task assignments as compared to those in [1–3] under different simulation scenarios. Specifically, we observe a performance improvement on the relative cost heuristic [3] by up to 61% in terms of minimizing the makespan of meta-tasks, which is quite significant.

The rest of the paper is organized as follows. Section 2 presents a formulation of stochastic scheduling problem together with two theorems which reveal the impact of ignoring the stochastic nature of task execution times on current scheduling heuristics. In section 3, we introduce our genetic algorithm. Section 4 proposes a way of computing the expected value of the maximum of machine loads, which is required by our genetic algorithm. We present our simulation results in section 5. Finally, section 6 gives our concluding remarks.

## 2. Stochastic scheduling problem

In this section, a formulation of the problem of stochastic scheduling of a meta-task is presented. In addition, it is shown that the optimization criterion on which current scheduling algorithms are based can result in inefficient task assignments.

A heterogeneous computing system is assumed to be composed of $p$ heterogeneous machines; let $M = \{m_1, m_2, \dots, m_p\}$ denote a set of heterogeneous machines. A set of $n$ tasks with no data dependencies is assumed to be scheduled on this system; let $V = \{v_1, v_2, \dots, v_n\}$ denote a set of independent tasks (a meta-task). Since task execution times are typically non-deterministic, the execution time of task $v_i$ on machine $m_j$ is represented by random variable $\tau_{i,j}$, $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant p$, where $E[\tau_{i,j}] = \bar{\tau}_{i,j}$ and $E[.]$ denotes the expected value of a random variable. In addition, random variables $\tau_{i,j}$ are assumed to be independent of each other simply because the execution times of a task on different machines and the execution times of different tasks are independent of each other.

Suppose that a task assignment denoted by $\mathcal{X}$ is given. The schedule length under task assignment $\mathcal{X}$ is defined to be:

$$T_F(\mathcal{X}) = \max_{m_j \in M} \left\{ \sum_{v_i \in V} x_{i,j} \tau_{i,j} \right\}, \tag{1}$$

where $x_{i,j}$ is a binary variable: $x_{i,j} = 1$ if and only if $v_i$ is assigned to $m_j$; $x_{i,j} = 0$, otherwise. The stochastic scheduling problem is defined to be:

$$\min_{\mathcal{X} \in \pi} \left\{ \bar{T}_F(\mathcal{X}) \right\}, \tag{2}$$

where $\bar{T}_F(\mathcal{X}) = E[T_F(\mathcal{X})]$ and $\pi = \{\mathcal{X} \mid \mathcal{X}$ is a valid task assignment$\}$ denotes all possible task assignments. The objective function given by (1) should be used in developing meta-task scheduling algorithms. Most of the existing scheduling algorithms, however, assume a different objective function from (1) simply because $\bar{T}_F(\mathcal{X})$ is difficult to compute. The complexity of computing $\bar{T}_F(\mathcal{X})$ comes from the fact that (1) random variables are not identically distributed, (2) the summation of two random variables in (1) corresponds to their convolution, which could be difficult to compute, and (3) finding the probability density function of the maximum of possibly many random variables in a closed form may not be manageable.

It is worthwhile to note that, in [8], $\bar{T}_F(\mathcal{X})$ is derived as follows under the simplifying assumption that task execution times are identically, exponentially distributed.

$$\bar{T}_F(\mathcal{X}) = \tau \left[ \frac{n}{p} + H(p) - 1 \right], \tag{3}$$

where $\tau = \bar{\tau}_{i,j}$, $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant p$, $H(p) = \log(p) + \gamma$ and $\gamma = 0.577215\ldots$ is Euler's constant.

Existing scheduling algorithms, including fast greedy and min-min [1] and relative cost [3], attempt to find a solution to the following problem.

$$\min_{\mathcal{X} \in \pi} \left\{ \hat{T}_F(\mathcal{X}) \right\}, \tag{4}$$

where

$$\hat{T}_F(\mathcal{X}) = \max_{m_j \in M} \left\{ \sum_{v_i \in V} x_{i,j} \bar{\tau}_{i,j} \right\}. \tag{5}$$

Note that $\hat{T}_F(\mathcal{X}) = \tau \lceil \frac{n}{p} \rceil$, which is different from $\bar{T}_F(\mathcal{X})$ in (3) under the same assumptions.

In the literature, several scheduling algorithms using (5) as the objective function to be minimized are shown to perform well provided that task execution times are deterministic. However, in the following, we prove that using objective function (5) may lead to poor task assignments if task execution times are not deterministic. It should be noted that the following two theorems were previously stated in [6]. Simpler proofs, however, are presented in this study.

**Theorem 1.** $\bar{T}_F(\mathcal{X}) \geqslant \hat{T}_F(\mathcal{X})$, that is, the expected value of actual schedule length $\bar{T}_F(\mathcal{X})$ is underestimated by $\hat{T}_F(\mathcal{X})$.

*Proof.* $T_F(\mathcal{X})$ is a nondecreasing and convex function of random variables $\tau_{i,j}$, since max is a nondecreasing and convex function. With respect to Jensen's inequality, the following is true,

$$E\big[f(Y)\big] \geqslant f\big(E[Y]\big),$$

where $Y$ is a random variable with finite mean and $f$ is a convex function. Using Jensen's inequality together with the linearity property of expected values,

$$\begin{aligned}\bar{T}_F(\mathcal{X}) &= E\left[\max_{m_j \in M}\left\{\sum_{v_i \in V} x_{i,j}\tau_{i,j}\right\}\right] \\ &\geqslant \max_{m_j \in M}\left\{E\left[\sum_{v_i \in V} x_{i,j}\tau_{i,j}\right]\right\} \\ &\geqslant \max_{m_j \in M}\left\{\sum_{v_i \in V} x_{i,j}E[\tau_{i,j}]\right\} \\ &\geqslant \hat{T}_F(\mathcal{X}).\end{aligned}$$

□

**Theorem 2.** Given that $\hat{\mathcal{X}}^*$ is an optimal task assignment of problem (4), the expected value of actual schedule length due to $\hat{\mathcal{X}}^*$ can be greater than the one due to $\mathcal{X}^*$, which is an optimal task assignment of problem (2).

*Proof.* For the proof of this theorem, it is sufficient to provide an example for which the statement made in theorem 2 holds.

Suppose that two tasks will run on two heterogeneous machines. The probability density functions of task execution times are assumed to be $f_{\tau_{1,1}}(t) = 0.01\mathrm{e}^{-0.01t}$, $f_{\tau_{1,2}}(t) = 0.02\mathrm{e}^{-0.02t}$, $f_{\tau_{2,1}}(t) = 0.02\mathrm{e}^{-0.02t}$, and $f_{\tau_{2,2}}(t) = 0.1\mathrm{e}^{-0.1t}$, where $t \geqslant 0$. For this scenario, table 2 shows the four possible task assignments and their corresponding $\bar{T}_F(\mathcal{X})$ and $\hat{T}_F(\mathcal{X})$.

According to table 2, optimal solutions of problem (2) and (4) are different and given by the fourth and third task assignments in the table, respectively. From the last column of the table, the optimal solution of problem (2) will result in a schedule length of 60 time units, whereas the optimal solution of problem (4) will lead to a schedule length of 75 time units. As a result, theorem 2 holds. □

Theorem 2 implies that an optimal solution with respect to (5) obtained by an exhaustive search may not be the actual optimal solution and thus may result in a schedule length arbitrarily larger than the actual optimal schedule length.

Table 2
$\bar{T}_F(\mathcal{X})$ and $\hat{T}_F(\mathcal{X})$ for the four possible task assignments. The optimal solution of problem (2) is $(v_1, m_2)$ and $(v_2, m_2)$, and the optimal solution of problem (4) is $(v_1, m_2)$, $(v_2, m_1)$.

| Task assignment | $\hat{T}_F(\mathcal{X})$ | $\bar{T}_F(\mathcal{X})$ |
|---|---|---|
| $(v_1, m_1)$, $(v_2, m_1)$ | 150 | 150 |
| $(v_1, m_1)$, $(v_2, m_2)$ | 100 | 101 |
| $(v_1, m_2)$, $(v_2, m_1)$ | 50 | 75 |
| $(v_1, m_2)$, $(v_2, m_2)$ | 60 | 60 |

```
1. initial_population_generation();
2. fitness_evaluation();
3. while (population is not converged)
4.     selection();
5.     crossover();
6.     mutation();
7.     fitness_evaluation();
8. endwhile.
```

Figure 1. The genetic algorithm implemented.

## 3. Meta-task scheduling by a genetic algorithm

In this section, a genetic algorithm for scheduling meta-tasks is presented. A genetic algorithm can inherently use different objective functions, one at a time, by modifying only the fitness evaluation phase. Thus, it is perfect to study the impact of using optimization criterion (1) instead of (5).

Genetic algorithms (GAs) [9] are heuristic search techniques that are founded upon the principle of evolution, i.e., *survival of the fittest*. Basically, any solution in the search space of an optimization problem is represented by a *chromosome*, where a set of chromosomes is referred to as a *population*. The quality of a chromosome is determined by a *fitness function*, which is the problem-dependent objective function to be optimized. Three genetic operators, namely *selection*, *crossover*, and *mutation*, are then applied one after the other to obtain a new population (generation) of chromosomes in which the expected quality over all the chromosomes is better than that of the previous generation. This process is repeated until the population is converged. Specifically, figure 1 shows the structure of the GA [10] implemented in this study and the details of each step is explained below.

### 3.1. Chromosome representation

A chromosome is basically a data structure into which a solution of the meta-task scheduling problem will be encoded. Determining this data structure is the first and very important step towards implementing an efficient genetic algorithm.

In this study, a chromosome is designed to be a list of $n$ elements where the $i$th element of the chromosome denotes the machine on which task $v_i$ is assigned. Let $\mathcal{C}_k = \{m_1^k, m_2^k, \ldots, m_n^k\}$ denote the $k$th chromosome in which task $v_i$ is scheduled to $m_i^k$.

### 3.2. Initial population generation

GAs use a population of chromosomes so as to search for an optimal solution starting from a number of initial solutions.

This set of initial solutions is referred to as the *initial population*. We let $N_p$ denote the number of chromosomes in the initial population, namely *population size*, and the population size is kept fixed through the generations of the population.

An initial population is randomly generated as follows: (1) A set of basis chromosomes is generated first. If the fitness of chromosomes is determined according to objective function (1), two basis chromosomes are generated by the fast greedy heuristic [1] and a random task assignment algorithm which assigns each task to a randomly chosen machine, respectively. This genetic algorithm will be referred to as the *stochastic genetic algorithm* (SGA). On the other hand, if objective function (5) is employed to compute the fitness of chromosomes, four basis chromosomes are generated by the fast greedy, min-min [1], relative cost [3], and random task assignment heuristics, respectively. This genetic algorithm will be referred to as the *deterministic genetic algorithm* (DGA). (2) A new chromosome other than basis chromosomes is generated: One of the previously generated chromosomes is randomly picked and mutated a random number of times (between one and the number of tasks) using the mutation operator defined later. (3) If the newly generated chromosome is identical to any of the previously generated ones, it is discarded. The process of chromosome generation is repeated until $N_p$ unique chromosomes are generated.

### 3.3. Fitness of a chromosome

The fitness value of a chromosome is computed according to $\bar{T}_F(\mathcal{X})$ in SGA and $\hat{T}_F(\mathcal{X})$ in DGA, as explained before. Thus, a chromosome with a small value of fitness is fitter than the one with a large value of fitness. Note that computing the actual schedule length $\bar{T}_F(\mathcal{X})$ is not trivial and will be addressed in section 4.

Both SGA and DGA incorporate *elitism* [11] as well. After the fitness evaluation process, the best chromosome of the current generation is found and compared against the elite chromosome, which is the fittest chromosome of the all previous generations and kept separately from the population. If the best chromosome is better than the elite chromosome, the best chromosome is copied to the elite chromosome. Otherwise, the worst chromosome in the population is replaced by the elite chromosome. Elitism is important because it assures that the quality of the best solution found over generations is monotonically increasing.

### 3.4. Selection

The selection operator is used to choose chromosomes from the current population to a *mating pool* where the fittest chromosome has the best chance of being selected, and so on. In the literature, several selection techniques, including proportionate, ranking, and tournament selection, have been proposed [12]. Since the ranking selection is shown to be a good choice in [12] in terms of growth ratio, takeover time, and time complexity, we choose the *ranking selection* as the selection operator.

In the implementation of the ranking selection, chromosomes are first sorted from best to worst with respect to their fitness values and each chromosome is assigned a rank between 1 and $N_p$, where the first ranked chromosome is the best chromosome. After that, the *roulette wheel sampling* scheme is used to choose chromosomes for the mating pool. In the roulette wheel sampling, a slot on the wheel is allocated to each chromosome where the size of the slot depends on the rank of the chromosome. Specifically, the angle of the slot allocated to the $i$th ranked chromosome, which is denoted by $A_i$, is defined to be [13]:

$$A_i = 360\frac{A-1}{A^{N_p}-1}A^{N_p-i}, \quad 1 \leqslant i \leqslant N_p, \tag{6}$$

where $A = A_i/A_{i+1}$, $1 \leqslant i < N_p$, is constant and is greater than one. Note that the first ranked chromosome will have the greatest slot, and so on. During the selection, a random number uniformly distributed between 0 and 360 is generated. This number falls in a slot on the roulette wheel and a copy of the chromosome associated with that slot is included in the mating pool. This process is repeated $N_p$ times. Note that it is possible for a chromosome to appear more than once in the mating pool. After the selection is over, chromosomes in the mating pool will be subject to crossover and mutation operators so as to form the next generation.

### 3.5. Crossover

The crossover operator is implemented as follows: (1) Chromosomes in the mating pool are randomly paired where a chromosome can be part of just one pair. (2) A pair of chromosomes, say $\mathcal{C}_i$ and $\mathcal{C}_j$, is taken from the mating pool. Each pair will be considered for crossover separately and only once. (3) The crossover operator is applied to the chosen pair with probability $\mu_c$, which is the *probability of crossover*. This parameter is experimentally determined. (4) For the pair, a cut-off point, which divides chromosomes $\mathcal{C}_i$ and $\mathcal{C}_j$ into top and bottom parts, is randomly generated. (5) Two new chromosomes are generated; the machine to which task $v_k$ in the bottom part of $\mathcal{C}_i$ ($\mathcal{C}_j$) is assigned is changed to the machine to which task $v_k$ is assigned in $\mathcal{C}_j$ ($\mathcal{C}_i$). These two new chromosomes are put back into the mating pool.

### 3.6. Mutation

The mutation operator is implemented as follows: (1) A chromosome is chosen from the mating pool. Each chromosome will be considered for mutation separately and only once. (2) The mutation operator is applied to the chosen chromosome with probability $\mu_m$, which is the *probability of mutation*. This parameter is experimentally determined. (3) A new chromosomes is generated; machine assignments of randomly chosen two tasks are swapped. This new chromosome is put back into the mating pool.

After mutating chromosomes in the mating pool, the chromosomes in the mating pool constitute the next generation.

## 4. Computation of $\bar{T}_F(\mathcal{X})$

In order to compute $\bar{T}_F(\mathcal{X})$, the probability density functions of random variables $\tau_{i,j}$, $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant p$, are re-

quired. These probability density functions may be estimated by using a technique such as the kernel density estimation method [14]. In this study, however, each random variable is associated with an approximate probability density function which has the same first and second moments as the corresponding exact probability density function. The rationale behind using approximate probability density functions is to take the variations in the execution times of tasks into account without dealing with the estimation of the exact ones while making scheduling decisions. Note that first and second moments can be easily determined by off-line execution of tasks several times on target machines. Furthermore, first moments (means) are also needed in existing scheduling algorithms.

In terms of approximating a probability density function, for the purposes of this study, any probability density function with a variable coefficient of variation, such as the Gamma distribution, can be assumed. However, in order for $\bar{T}_F(\mathcal{X})$ to be computed relatively easily and for the sake of analytical tractability, the *branching Erlang* distribution [15] is embraced. That is, the exact probability density function of $\tau_{i,j}$ is approximated by a branching Erlang distribution with the same first two moments as the exact one. Note that the same technique is also used in the analysis of Markovian queuing systems if the service time distribution of a server (or the inter-arrival time distribution of customers) is not an exponential distribution.

Let $Y$ be a random variable and $f_Y(t)$ be its probability density function. Let $\sigma_Y^2$ and $\upsilon_Y = \frac{\sigma_Y}{E[Y]}$ denote the variance and coefficient of variation of random variable $Y$, respectively. For the given values of $E[Y]$ and $\upsilon_Y$ of an exact probability density function, following branching Erlang distributions ($f_Y(t)$) are used to approximate it. If $\upsilon_Y \leqslant 1$, $f_Y(t)$ is assumed to be:

$$f_Y(t) = a\lambda \mathrm{e}^{-\lambda t} + \frac{(1-a)\lambda^k}{(k-1)!} t^{k-1} \mathrm{e}^{-\lambda t}, \tag{7}$$

where

$$k = \left\lceil \frac{1}{\upsilon_Y^2} \right\rceil, \quad a = \frac{2k\upsilon_Y^2 + k - 2 - \sqrt{k^2 + 4 - 4k\upsilon_Y^2}}{2(k-1)(\upsilon_Y^2 + 1)},$$
$$\lambda = \frac{k - a(k-1)}{E[Y]},$$

and $t \geqslant 0$. $f_Y(t)$ is a function of $k$, $a$, and $\lambda$ ($f_Y(t) = f_1(k, a, \lambda)$). If $\upsilon_Y > 1$, $f_Y(t)$ is assumed to be:

$$f_Y(t) = a\lambda_1 \mathrm{e}^{-\lambda_1 t} + \frac{(1-a)\lambda_1\lambda_2}{\lambda_2 - \lambda_1}\left(\mathrm{e}^{-\lambda_1 t} - \mathrm{e}^{-\lambda_2 t}\right), \tag{8}$$

where

$$a = \upsilon_Y^2\left(1 - \sqrt{1 - \frac{2}{1+\upsilon_Y^2}}\right),$$
$$\lambda_1 = \frac{1 + \sqrt{1 - \frac{2}{1+\upsilon_Y^2}}}{E[Y]},$$
$$\lambda_2 = \frac{1 - \sqrt{1 - \frac{2}{1+\upsilon_Y^2}}}{E[Y]},$$

and $t \geqslant 0$. $f_Y(t)$ is a function of $a$, $\lambda_1$, and $\lambda_2$ ($f_Y(t) = f_2(a, \lambda_1, \lambda_2)$).

Assuming (7) or (8) as the probability density functions of task execution times, the approximate probability density function of random variable $T_F(\mathcal{X})$ must be computed next.

$$\begin{aligned} T_F(\mathcal{X}) &= \max_{m_j \in M}\left\{\sum_{v_i \in V} x_{i,j}\tau_{i,j}\right\} \\ &= \max\{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_p\}, \\ &= \max\{\max\{\mathcal{Y}_1, \mathcal{Y}_2\}, \mathcal{Y}_3, \ldots, \mathcal{Y}_p\} \\ &= \max\{\mathcal{U}_2, \mathcal{Y}_3, \ldots, \mathcal{Y}_p\} \\ &\;\;\vdots \\ &= \max\{\mathcal{U}_{p-1}, \mathcal{Y}_p\} = \mathcal{U}_p, \end{aligned} \tag{9}$$

where $\mathcal{Y}_l = \sum_{v_i \in V} x_{i,l}\tau_{i,l}$, $1 \leqslant l \leqslant p$, is the total load on machine $m_l$ and $\mathcal{U}_l = \max\{\mathcal{U}_{l-1}, \mathcal{Y}_l\}$, $\mathcal{U}_1 = \mathcal{Y}_1$ and $2 \leqslant l \leqslant p$, is the maximum of machine loads from the first machine to the $l$th machine.

The computation of approximate probability density function $f_{\mathcal{Y}_l}(t)$ of random variable $\mathcal{Y}_l$, $1 \leqslant l \leqslant p$, can be done as follows. Since random variables ($\tau_{i,j}$) are assumed to be independent of each other,

$$E[\mathcal{Y}_l] = \sum_{v_i \in V} x_{i,l}\bar{\tau}_{i,l} \quad \text{and} \quad \sigma_{\mathcal{Y}_l}^2 = \sum_{v_i \in V} x_{i,l}\sigma_{\tau_{i,l}}^2.$$

After the expected value and variance of $\mathcal{Y}_l$ are computed, $\upsilon_{\mathcal{Y}_l}$ is computed. Based on the value of $\upsilon_{\mathcal{Y}_l}$, either (7) or (8) is assumed to be $f_{\mathcal{Y}_l}(t)$.

From probability theory [16], the probability density function of random variable $\mathcal{U}_l = \max\{\mathcal{U}_{l-1}, \mathcal{Y}_l\}$, $2 \leqslant l \leqslant p$, can be computed as

$$f_{\mathcal{U}_l}(t) = f_{\mathcal{Y}_l}(t)F_{\mathcal{U}_{l-1}}(t) + F_{\mathcal{Y}_l}(t)f_{\mathcal{U}_{l-1}}(t),$$

where $F_{\mathcal{Y}_l}(t) = \int_0^t f_{\mathcal{Y}_l}(t)\,\mathrm{d}t$ and $F_{\mathcal{U}_{l-1}}(t) = \int_0^t f_{\mathcal{U}_{l-1}}(t)\,\mathrm{d}t$. Once $f_{\mathcal{U}_l}(t)$ is computed, its expected value and coefficient of variation are computed. With respect to its value of coefficient of variation, either (7) or (8) is assumed to be $f_{\mathcal{U}_l}(t)$. Note that, since $f_{\mathcal{Y}_l}(t)$ and $f_{\mathcal{U}_{l-1}}(t)$ can be either (7) or (8), there are four possible ways to express $f_{\mathcal{U}_l}(t)$, each of which is derived in appendix A.

In order to compute approximate probability density function $f_{T_F(\mathcal{X})}(t)$ of random variable $T_F(\mathcal{X})$, note that $T_F(\mathcal{X}) = \mathcal{U}_p$. That is, $f_{T_F(\mathcal{X})}(t)$ can be found by applying the max function $(p-1)$ times. After $f_{T_F(\mathcal{X})}(t)$ is found, the expected value of $T_F(\mathcal{X})$ is computed as $E[T_F(\mathcal{X})] = \int_0^t t f_{T_F(\mathcal{X})}(t)\,\mathrm{d}t$.

## 5. Experiments

Our goal in the experiments is to study if using $\bar{T}_F(\mathcal{X})$ instead of $\hat{T}_F(\mathcal{X})$ while making scheduling decisions will make a difference in terms of minimizing the schedule length of meta-tasks.

In the experiments, the number of machines is set to either 25 or 50 and the number of tasks ranges from 50 to 250

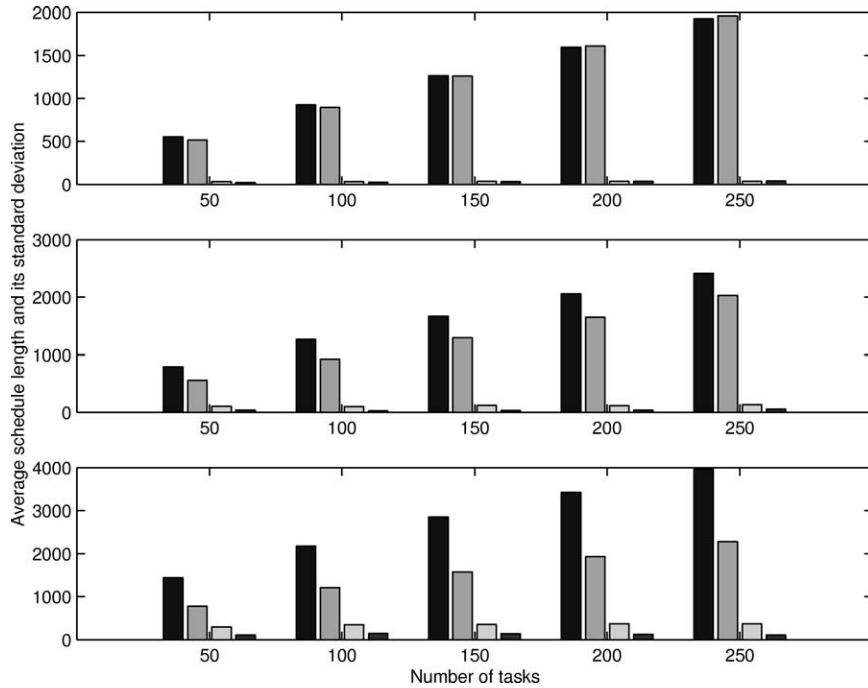


Figure 2. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for low task, low machine heterogeneity under RC and SGA for $p = 25$.

with the increments of 50. The expected execution times of tasks on machines are generated by using the method proposed in [17] and they are stored in an $n$ by $p$ *ETC* matrix, where $ETC_{i,j} = \bar{\tau}_{i,j}$, $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant p$. For all ETC matrices generated, $\mu_{\text{task}} = 100$, $\mu_{\text{mach}} = 100$, $V_{\text{task}} = 0.25$ and 1.0 for low and high task heterogeneity, respectively, and $V_{\text{mach}} = 0.25$ and 1.0 for low and high machine heterogeneity, respectively, where $\mu_{\text{task}}$, $\mu_{\text{mach}}$, $V_{\text{task}}$, and $V_{\text{mach}}$ are the parameters defined in [17]. In each experiment, in addition to *ETC* matrix, an $n$ by $p$ *ECV* matrix is generated as well, where $ECV_{i,j} = \upsilon_{\tau_{i,j}}$, $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant p$. As a result, an approximate probability density function is well-defined for each random variable $\tau_{i,j}$ with respect to $(ETC_{i,j}, ECV_{i,j})$. Furthermore, *ECV* matrix is generated in such a way that we can model different computing platforms including the ones where actual task execution times are close to their expected values or they significantly vary from their expected values. In this study, we consider only three distinct computing platforms with respect to *ECV*: In the first computing platform, $ECV_{i,j} \leqslant 1$ is assumed for each task-machine pair. In the second computing platform, 75% of task-machine pairs are assumed to have low coefficient of variations ($ECV_{i,j} \leqslant 1$) and 25% of them have high coefficient of variations ($1 < ECV_{i,j} \leqslant 2.5$). Finally, in the third computing platform, 50% of task-machine pairs are assumed to have low coefficient of variations and 50% of them have very high coefficient of variations ($1 < ECV_{i,j} \leqslant 5$). The first, second, and third computing platforms will be referred to as the system with low, medium, and high variability, respectively.

With respect to the genetic algorithm presented, $N_p$ is taken as 100 and $A$, $\mu_c$, and $\mu_m$ are set to 1.1, 0.6, and 0.2, respectively. The genetic algorithm is stopped if the number of generations has reached to 1000, or if the elite chromosome has not changed in the last 100 iterations.

In our experiments, we perform two distinct comparisons, namely SGA versus relative cost (RC) [3] and SGA versus DGA. During the comparisons, *ETC* and *ECV* matrices are first generated. Then, the corresponding heuristic is run to obtain a task assignment. Note that SGA needs both *ETC* and *ECV* to work, while RC and DGA need only *ETC*. Finally, the makespan of the task assignment output by the heuristic is find out by generating a simulated actual execution time for each task on its assigned machine using the Gamma distribution. This last step is repeated 25 times, keeping the task assignment fixed, and the average makespan over these repetitions is taken as the makespan of the task assignment. The Gamma distribution is used for simulated actual execution times due to the fact that the Gamma distribution can approximate other well-known distributions, including exponential, Erlang, and normal (without the negative values) distributions, and achieve any specified value of the coefficient of variation with the proper choice of its parameters. Thus, assuming the Gamma distribution increases the possibility that the simulated random variables could be synthesized to closely model some real life computing environments.

### *5.1. SGA versus RC*

In this section, we present our simulation results for the comparison of SGA with RC, since RC is the best meta-task scheduling heuristic in the literature to the best knowledge of authors. These results are shown in figures 2–9, where (1) each set of three figures, from top to bottom, shows the re-

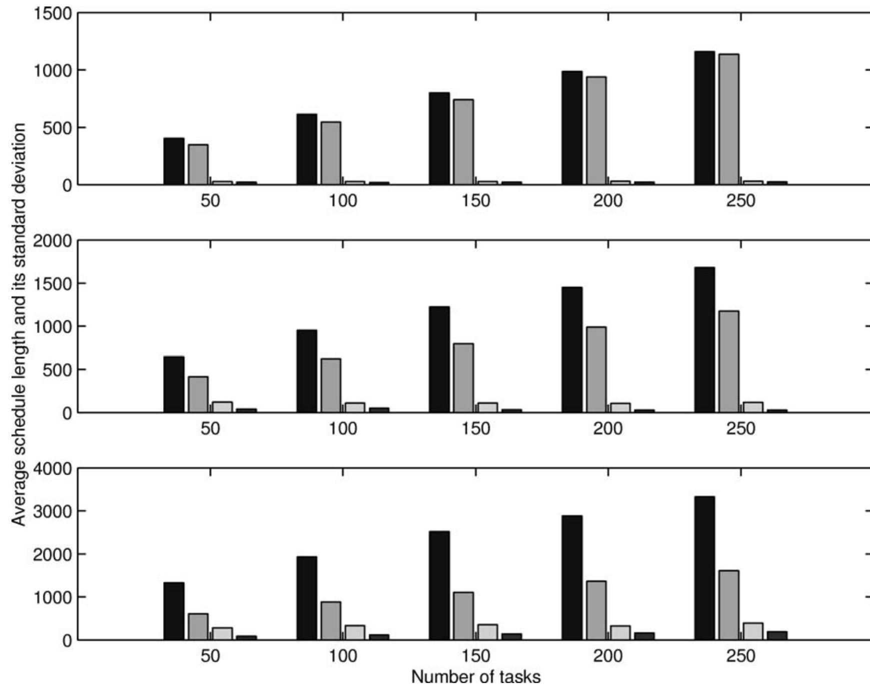


Figure 3. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for low task, low machine heterogeneity under RC and SGA for $p = 50$.

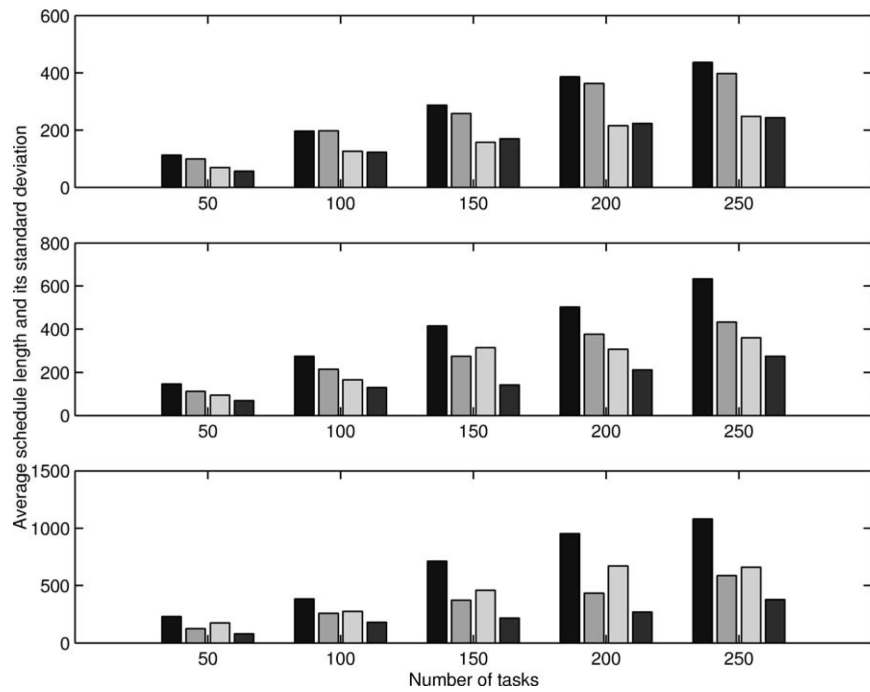


Figure 4. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for low task, high machine heterogeneity under RC and SGA for $p = 25$.

sults for a system with low variability, a system with medium variability, and a system with high variability, respectively, (2) each set of four bars, from left to right, corresponds to the average schedule length of the RC, the average schedule length of the SGA, the standard deviation of the schedule length under the RC, and the standard deviation of the schedule length under the SGA, respectively, and (3) each data point is the average of the data obtained in 100 experiments.

Based on the simulation results in figures 2–9, we form two tables, 3 and 4, in order to summarize our findings on the performance of the SGA and the RC in terms of minimizing the schedule length of meta-tasks. Specifically, tables 3 and 4 present the mean and maximum percentage of the decrease in

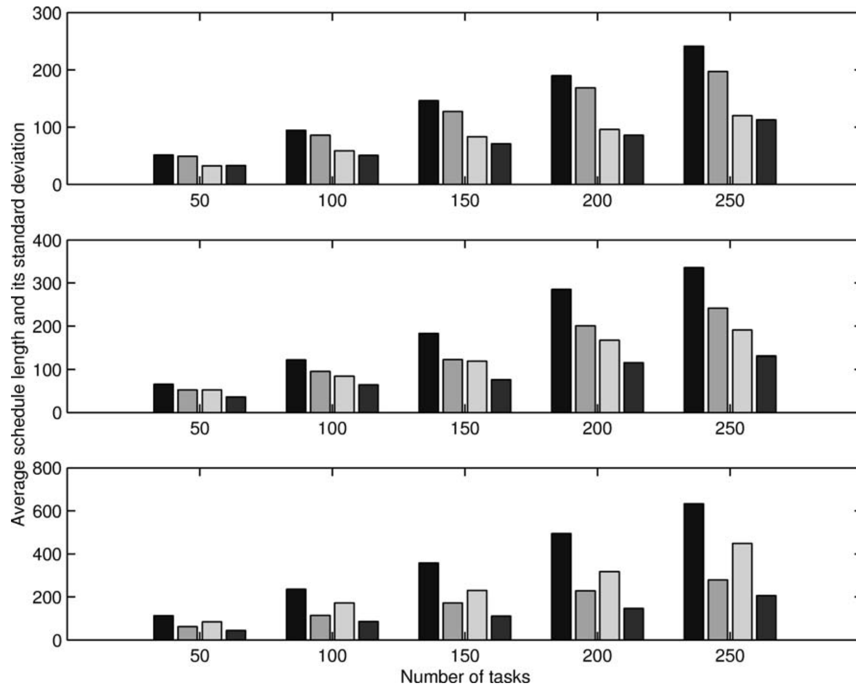


Figure 5. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for low task, high machine heterogeneity under RC and SGA for $p = 50$.

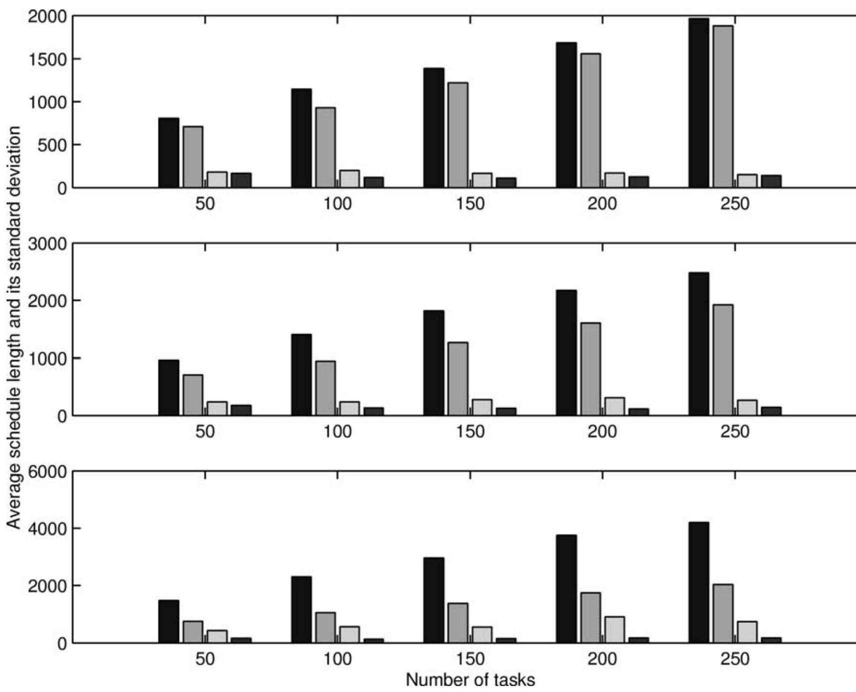


Figure 6. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for high task, low machine heterogeneity under RC and SGA for $p = 25$.

the schedule length for SGA over RC. Note that, in the tables (low, high) means low task heterogeneity, high machine heterogeneity and so on, and Low, Medium, and High means a system with low variability, a system with medium variability, and a system with high variability, respectively.

According to tables 3 and 4, it is evident that SGA outperforms RC in terms of minimizing the schedule length. Specifically, SGA decreases the average schedule length by up to 61%. Knowing that the RC heuristic is the best among classic heuristics, the performance of SGA seems to be outstanding. We may attribute this performance gain to the following facts. First, genetic algorithms are proved to be very efficient in solving many hard optimization problems including the task assignment problem and they most of the time produce bet-

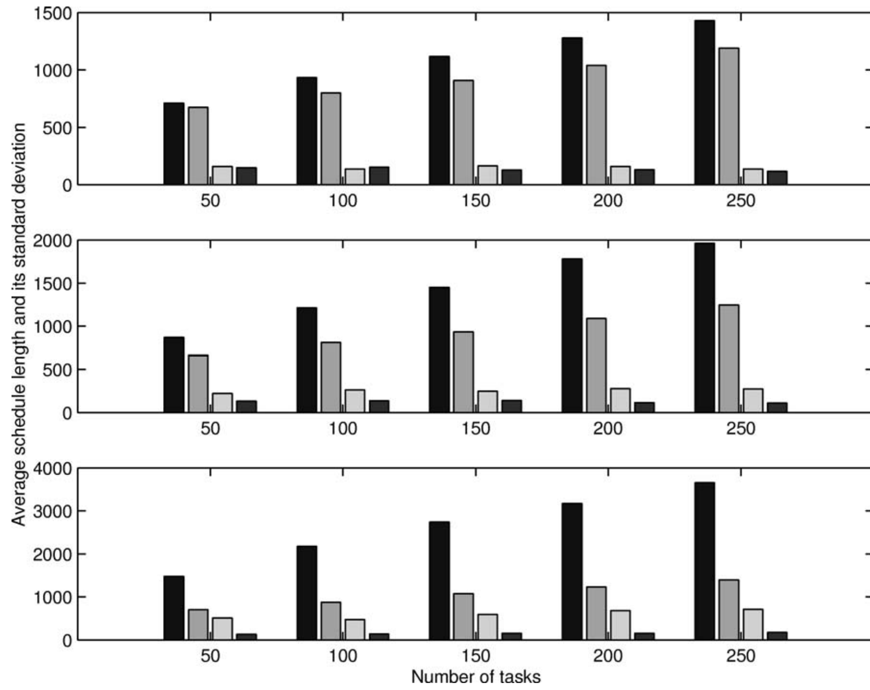


Figure 7. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for high task, low machine heterogeneity under RC and SGA for $p = 50$.

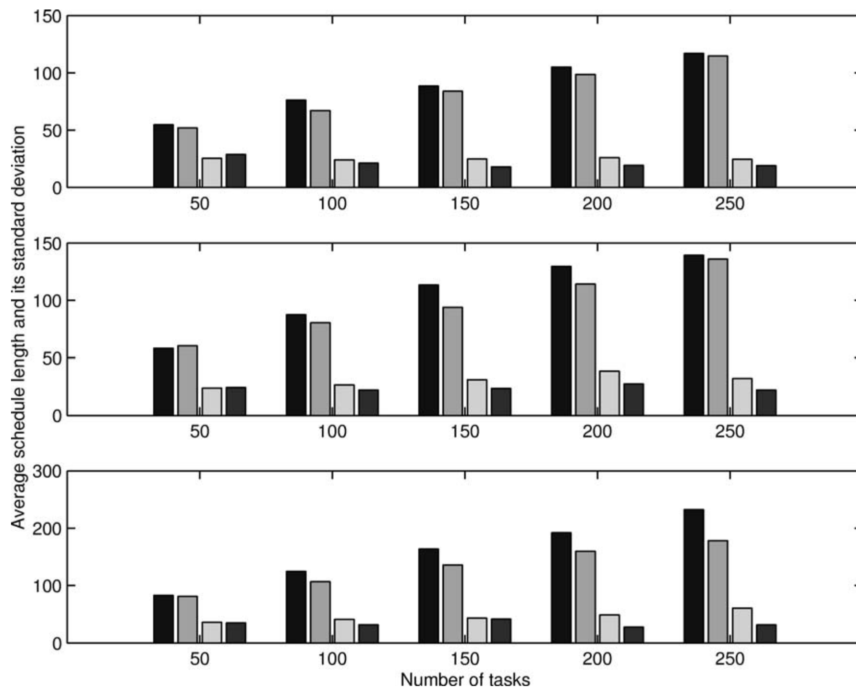


Figure 8. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for high task, high machine heterogeneity under RC and SGA for $p = 25$.

ter results than classic approaches. Second, SGA has a way of accounting for the variances in task execution times. This allows SGA to have more information on the characteristics of both the meta-task and the computing environment, which eventually leads to superior performance.

From the tables, we have also observed the following trends. While the variability of the system increases, the amount of decrease in the average schedule length under SGA increases. The reason for this phenomenon is as follows. With the increasing variability of the system, RC bases its scheduling decisions on the mean task execution times which become more and more irrelevant to the actual task execution times. Thus, RC produces poor task assignments if the system variability is high. On the other hand, SGA is able to perform

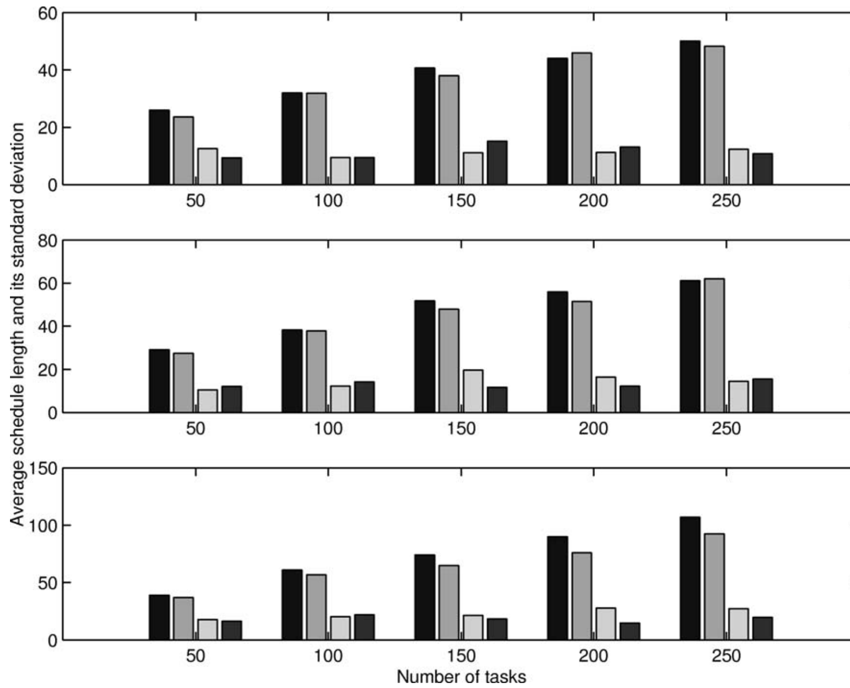


Figure 9. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for high task, high machine heterogeneity under RC and SGA for $p = 50$.

Table 3
The percentage of the decrease in the schedule length for SGA over RC for $p = 25$. The table shows that SGA outperforms RC for four distinct (task, machine) heterogeneity and three different system variability.

| | Low | | Medium | | High | |
|---|---|---|---|---|---|---|
| | Mean | Max | Mean | Max | Mean | Max |
| (low, low) | 2 | 7 | 23 | 30 | 44 | 46 |
| (low, high) | 7 | 12 | 27 | 34 | 45 | 55 |
| (high, low) | 11 | 19 | 28 | 33 | 53 | 54 |
| (high, high) | 6 | 11 | 7 | 17 | 15 | 23 |

Table 4
The percentage of the decrease in the schedule length for SGA over RC for $p = 50$. The table indicates that SGA outperforms RC for four distinct (task, machine) heterogeneity and three different system variability.

| | Low | | Medium | | High | |
|---|---|---|---|---|---|---|
| | Mean | Max | Mean | Max | Mean | Max |
| (low, low) | 8 | 14 | 33 | 36 | 54 | 56 |
| (low, high) | 11 | 18 | 26 | 33 | 52 | 56 |
| (high, low) | 15 | 19 | 34 | 40 | 59 | 61 |
| (high, high) | 3 | 9 | 4 | 8 | 11 | 15 |

well even if the system variability is high, thanks to its design. The next observation is that increasing the number of machines from 25 to 50 has helped SGA to widen the performance gap between itself and RC if (task, machine) heterogeneity is (low, low), (low, high), and (high, low). This is mainly due to the fact that increasing the number of machines lowers the performance of RC more as compared to SGA. We have an opposite case if (task, machine) heterogeneity is (high, high), i.e., the performance gap between SGA and RC is narrowing.

Another important performance objective is the standard deviation of the schedule length. We observe that SGA most of the time results in a lower standard deviation for the schedule length as compared to RC. A crucial implication of this result is the fact that the performance of meta-tasks under the SGA is more predictable. Furthermore, we may attribute this result to the inability of the RC in taking the variances in the task execution times into account.

### 5.2. SGA versus DGA

This section shows the performance comparison between SGA and DGA. It is worth to remind that SGA needs to compute $\bar{T}_F(\mathcal{X})$ to assign a fitness value to a chromosome, while DGA evaluates $\hat{T}_F(\mathcal{X})$ using the expected values of execution times. In addition, since the initial population of DGA includes a chromosome into which the task assignment produced by RC is encoded, it is expected that DGA will compete better than RC with SGA.

The simulation results are shown in figures 10–13 for $p = 50$. Based on the results in these figures, we form table 5 which summarizes our findings on the performance of the SGA and the DGA in terms of minimizing the schedule length of meta-tasks. Table 5 clearly indicates that SGA outperforms DGA in terms of minimizing the schedule length. Specifically, SGA decreases the average schedule length by up to 65%. This performance gain is again due to the fact that SGA accounts for the possible variations in the task execution times when making scheduling decisions. From figures 10–13, we have also noted that SGA minimizes the standard deviation of the schedule length.

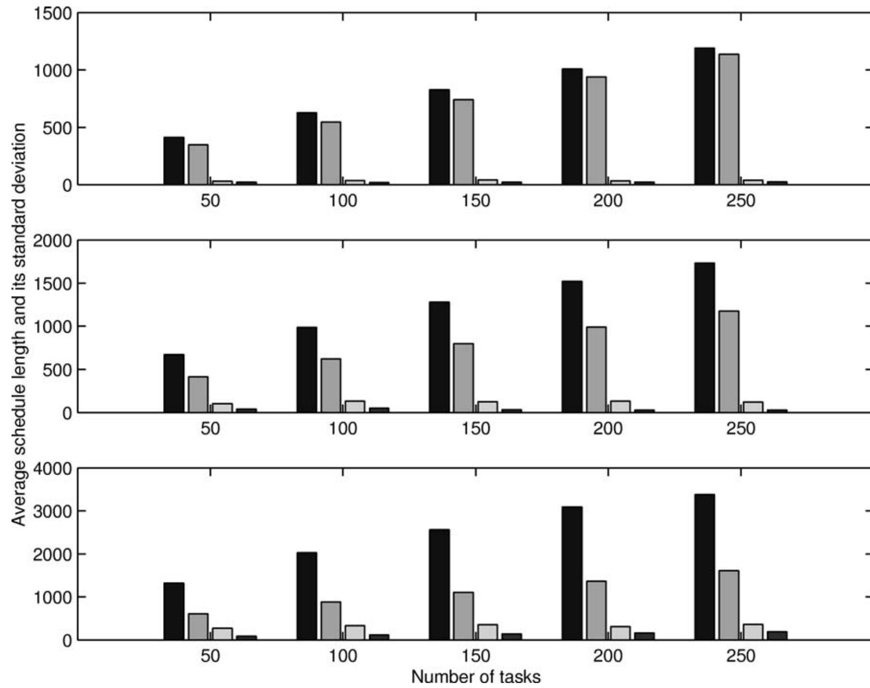


Figure 10. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for low task, low machine heterogeneity under DGA and SGA.

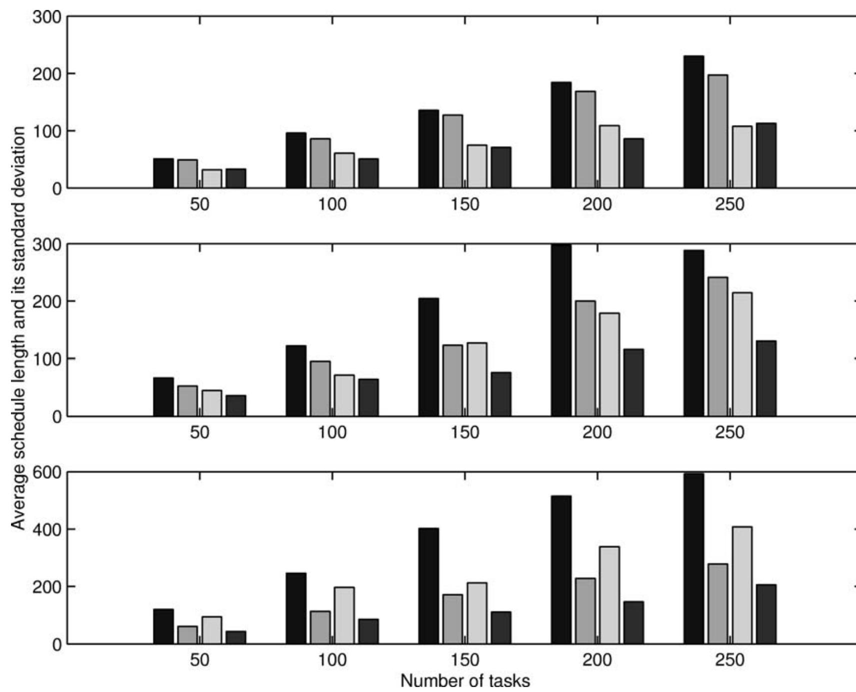


Figure 11. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for low task, high machine heterogeneity under DGA and SGA.

When we compare table 5 with table 4, we surprisingly observe that the RC produces better task assignments than the DGA. As far as this result is concerned, one may argue that DGA is supposed to improve on RC. However, this would be true if the actual execution times of tasks were equal to the expected ones. In order to justify our argument, we compare DGA with RC when the actual execution times are the same as the expected ones and we summarize the results in table 6. As expected, DGA improves on RC by up to 29%.

## 6. Conclusions

This study has focused on revealing the fact that existing scheduling algorithms whose objective is to minimize the maximum of the expected values of machine loads may produce inefficient schedules in a computing system where the execution times of tasks are stochastic. This fact has been proved theoretically and also supported by an extensive set of experiments. As shown in the previous section, using a

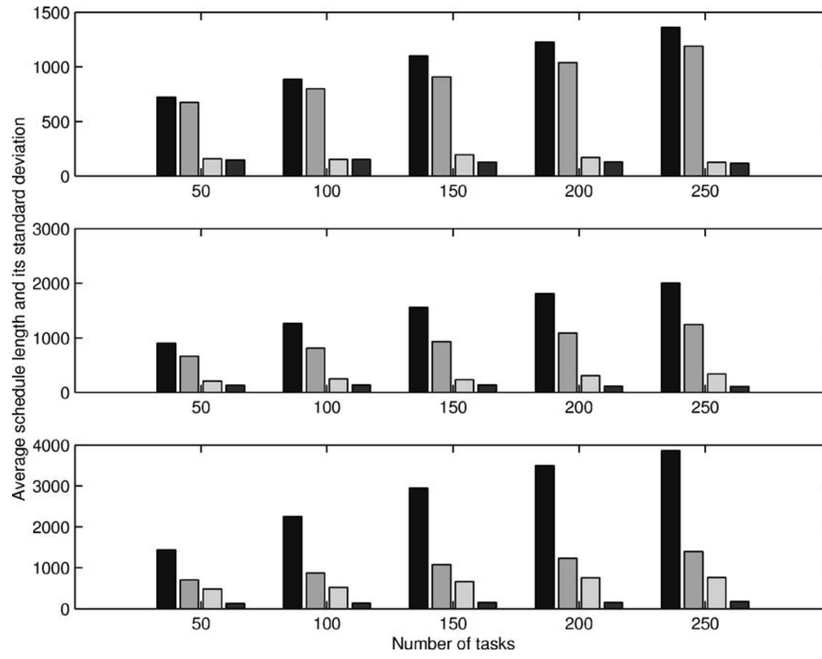


Figure 12. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for high task, low machine heterogeneity under DGA and SGA.

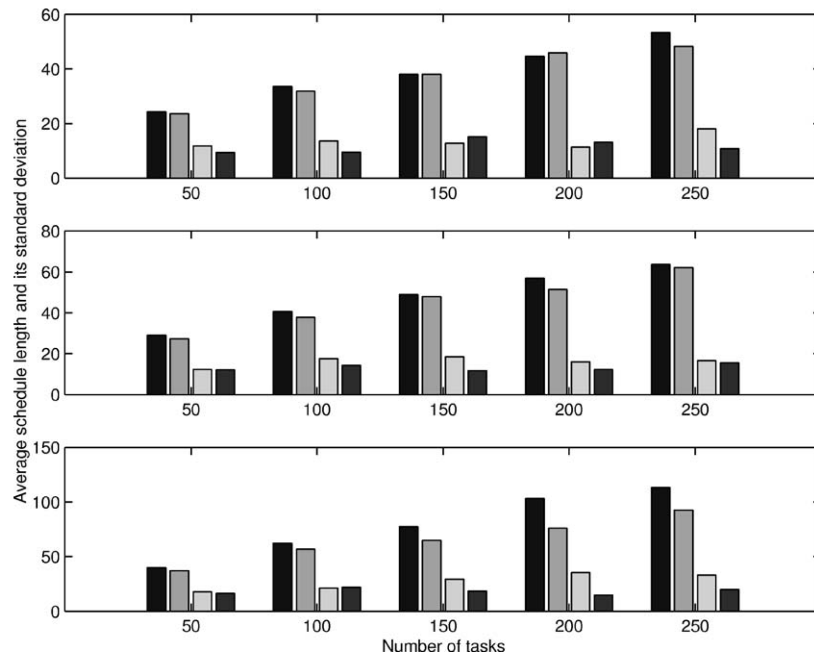


Figure 13. Average schedule lengths of meta-tasks and standard deviation of schedule lengths for high task, high machine heterogeneity under DGA and SGA.

Table 5
The percentage of the decrease in the schedule length for SGA over DGA. According to the table, SGA outperforms DGA for four distinct (task, machine) heterogeneity and three different system variability.

| | Low | | Medium | | High | |
|---|---|---|---|---|---|---|
| | Mean | Max | Mean | Max | Mean | Max |
| (low, low) | 10 | 16 | 36 | 38 | 55 | 57 |
| (low, high) | 9 | 15 | 26 | 40 | 54 | 57 |
| (high, low) | 12 | 17 | 36 | 40 | 61 | 65 |
| (high, high) | 3 | 10 | 5 | 10 | 15 | 26 |

Table 6
Average schedule lengths and the percentage of the decrease in the schedule length for DGA over RC. According to the table, DGA improves on RC for four distinct (task, machine) heterogeneity provided that the actual execution times are the same as the expected ones.

| | RC | DGA | Improvement | |
|---|---|---|---|---|
| | | | Mean | Max |
| (low, low) | 425 | 392 | 9 | 15 |
| (low, high) | 738 | 714 | 3 | 4 |
| (high, low) | 700 | 552 | 19 | 29 |
| (high, high) | 274 | 250 | 8 | 10 |

genetic algorithm designed to account for the stochastic nature of task execution times makes it possible to obtain performance improvements in the schedule length of meta-tasks by up to 61%. Furthermore, the proposed genetic algorithm is shown to decrease the standard deviation of the schedule length. A challenge in devising stochastic scheduling algorithms, however, is the computation of $\bar{T}_F(\mathcal{X})$, where different probability density functions can be assumed for representing task execution times. In order to address this problem, a way of computing $\bar{T}_F(\mathcal{X})$ based on the Erlang branching distribution is developed.

According to [18], the width of a confidence interval $[t_{lb}, t_{ub}]$ for the running time of a task on a time-shared machine is a linear function of the task's nominal running time (running time on a space-shared machine), where $t_{lb}$ and $t_{ub}$ are the lower and upper bounds of the confidence interval computed to the task's requested confidence level for the running time. In other words, the longer the task's nominal running time is the more uncertain the task's expected execution time is on a time-shared machine. As a result, long-running tasks on a time-shared distributed computing system will be adversely affected by this uncertainty if such tasks are scheduled using heuristics similar to existing ones. On the other hand, as we show in the previous section, one can always implement the proposed SGA based on $\bar{\tau}_{i,j}$, $t_{lb}$, and $t_{ub}$ in order to produce high quality task assignments under uncertainty, which further emphasizes the importance of the proposed approach in this study.

## Appendix A. Computing an approximate PDF for the maximum of two random variables

Let the coefficients of variations of both $f_{\mathcal{Y}_l}(t)$ and $f_{\mathcal{U}_{l-1}}(t)$ be equal or less than one, that is, $\upsilon_{\mathcal{Y}_l} \leqslant 1$ and $\upsilon_{\mathcal{U}_{l-1}} \leqslant 1$. Consequently, $f_{\mathcal{Y}_l}(t) = f_1(k, a, \lambda)$ and $f_{\mathcal{U}_{l-1}}(t) = f_1(l, b, \zeta)$, and $f_{\mathcal{U}_l}(t)$ is found to be:

$$
\begin{aligned}
f_{\mathcal{U}_l}(t) = {} & a\lambda \mathrm{e}^{-\lambda t} + b\zeta \mathrm{e}^{-\zeta t} - ab(\lambda+\zeta)\mathrm{e}^{-(\lambda+\zeta)t} \\
& + (1-a)\frac{\lambda^k}{(k-1)!} t^{k-1}\left[\mathrm{e}^{-\lambda t} - b\mathrm{e}^{-(\lambda+\zeta)t}\right] \\
& + (1-b)\frac{\zeta^l}{(l-1)!} t^{l-1}\left[\mathrm{e}^{-\zeta t} - a\mathrm{e}^{-(\lambda+\zeta)t}\right] \\
& - (1-b)\left[a\lambda \sum_{i=0}^{l-1}\frac{\zeta^i t^i}{i!} + (1-a)\frac{\lambda^k}{(k-1)!} \times \sum_{i=0}^{l-1}\frac{\zeta^i t^{i+k-1}}{i!}\right]\mathrm{e}^{-(\lambda+\zeta)t} \\
& - (1-a)\left[b\zeta \sum_{i=0}^{k-1}\frac{\lambda^i t^i}{i!} + (1-b)\frac{\zeta^l}{(l-1)!} \times \sum_{i=0}^{k-1}\frac{\lambda^i t^{i+l-1}}{i!}\right]\mathrm{e}^{-(\lambda+\zeta)t}. \qquad \text{(A.1)}
\end{aligned}
$$

If $\upsilon_{\mathcal{Y}_l} \leqslant 1$ and $\upsilon_{\mathcal{U}_{l-1}} > 1$, $f_{\mathcal{Y}_l}(t) = f_1(k, a, \lambda)$ and $f_{\mathcal{U}_{l-1}}(t) = f_2(b, \zeta_1, \zeta_2)$, and $f_{\mathcal{U}_l}(t)$ is found to be:

$$
\begin{aligned}
f_{\mathcal{U}_l}(t) = {} & a\lambda \mathrm{e}^{-\lambda t} + \frac{\zeta_2 - b\zeta_1}{\zeta_2 - \zeta_1}\zeta_1 \mathrm{e}^{-\zeta_1 t} - \frac{\zeta_1 - b\zeta_1}{\zeta_2 - \zeta_1}\zeta_2 \mathrm{e}^{-\zeta_1 t} \\
& - \frac{a(\zeta_2 - b\zeta_1)}{\zeta_2 - \zeta_1}(\lambda+\zeta_1)\mathrm{e}^{-(\lambda+\zeta_1)t} \\
& + \frac{a(\zeta_1 - b\zeta_1)}{\zeta_2 - \zeta_1}(\lambda+\zeta_2)\mathrm{e}^{-(\lambda+\zeta_2)t} \\
& + (1-a)\frac{\lambda^k}{(k-1)!} t^{k-1}\mathrm{e}^{-\lambda t} \\
& + \frac{(1-a)}{\zeta_2 - \zeta_1}\frac{\lambda^k}{(k-1)!} t^{k-1} \\
& \times \left[(\zeta_1 - b\zeta_1)\mathrm{e}^{-(\lambda+\zeta_2)t} - (\zeta_2 - b\zeta_1)\mathrm{e}^{-(\lambda+\zeta_1)t}\right] \\
& - \frac{(\zeta_1 - a\zeta_1)(\zeta_2 - b\zeta_1)}{\zeta_2 - \zeta_1}\sum_{i=0}^{k-1}\frac{\lambda^i t^i}{i!}\mathrm{e}^{-(\lambda+\zeta_1)t} \\
& + \frac{(\zeta_1 - b\zeta_1)(\zeta_2 - a\zeta_1)}{\zeta_2 - \zeta_1}\sum_{i=0}^{k-1}\frac{\lambda^i t^i}{i!}\mathrm{e}^{-(\lambda+\zeta_2)t}. \qquad \text{(A.2)}
\end{aligned}
$$

If $\upsilon_{\mathcal{Y}_l} > 1$ and $\upsilon_{\mathcal{U}_{l-1}} \leqslant 1$, $f_{\mathcal{Y}_l}(t) = f_2(b, \zeta_1, \zeta_2)$ and $f_{\mathcal{U}_{l-1}}(t) = f_2(k, a, \lambda)$, and $f_{\mathcal{U}_l}(t)$ will be given by (A.2).

If $\upsilon_{\mathcal{Y}_l} > 1$ and $\upsilon_{\mathcal{U}_{l-1}} > 1$, $f_{\mathcal{Y}_l}(t) = f_2(a, \lambda_1, \lambda_2)$ and $f_{\mathcal{U}_{l-1}}(t) = f_2(b, \zeta_1, \zeta_2)$, and $f_{\mathcal{U}_l}(t)$ is found to be:

$$
\begin{aligned}
f_{\mathcal{U}_l}(t) = {} & \frac{1}{\lambda_2 - \lambda_1}\left[(\lambda_2 - a\lambda_1)\lambda_1 \mathrm{e}^{-\lambda_1 t} - (\lambda_1 - a\lambda_1)\lambda_2 \mathrm{e}^{-\lambda_2 t}\right] \\
& + \frac{1}{\zeta_2 - \zeta_1}\left[(\zeta_2 - b\zeta_1)\zeta_1 \mathrm{e}^{-\zeta_1 t} - (\zeta_1 - a\zeta_1)\zeta_2 \mathrm{e}^{-\zeta_2 t}\right] \\
& - \frac{1}{(\lambda_2 - \lambda_1)(\zeta_2 - \zeta_1)} \\
& \times \Big[\left[\lambda_2\zeta_2 - a\lambda_1\zeta_2 - b\lambda_2\zeta_1 + ab(\lambda_1\zeta_1 - 2\lambda_2\zeta_2)\right] \\
& \times (\lambda_1 + \zeta_1)\mathrm{e}^{-(\lambda_1+\zeta_1)t} \\
& - (1-b)(\lambda_2 - a\lambda_1)(\lambda_1 + \zeta_2)\zeta_1 \mathrm{e}^{-(\lambda_1+\zeta_2)t} \\
& - (1-a)(\zeta_2 - b\zeta_1)(\lambda_2 + \zeta_1)\lambda_1 \mathrm{e}^{-(\lambda_2+\zeta_1)t} \\
& + (1-a)(1-b)(\lambda_2 + \zeta_2)\lambda_1\zeta_1 \mathrm{e}^{-(\lambda_2+\zeta_2)t}\Big]. \qquad \text{(A.3)}
\end{aligned}
$$

## References


[1] T.D. Braun et al., A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems, in: *IPPS/SPDP Workshop on Heterogeneous Computing*, San Juan, Puerto Rico (April 1999) pp. 15–29.

[2] M.-Y. Wu, W. Shu and H. Zhang, Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems, in: *IPDPS Workshop on Heterogeneous Computing*, Cancún, Mexico (May 2000) pp. 375–385.

[3] M.-Y. Wu and W. Shu, A high-performance mapping algorithm for heterogeneous computing systems, in: *International Parallel and Distributed Processing Symposium*, San Francisco, CA (April 2001).

[4] R. Armstrong, D. Hensgen and T. Kidd, The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions, in: *IPPS/SPDP Workshop on Heterogeneous Computing*, Orlando, FL (March 1998).

[5] S.M. Figueira and F. Berman, A slowdown model for applications executing on time-shared clusters of workstations, IEEE Trans. Parallel and Distributed Systems 12 (June 2001) 653–670.

[6] T. Kidd and D. Hensgen, Why the mean is inadequate for accurate scheduling decisions, in: *The Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms, and Networks*, Perth, Australia (June 1999).
[7] M.H. Balter, M.E. Crovella and C.D. Murta, On choosing a task assignment policy for a distributed server system, Journal of Parallel and Distributed Computing 59 (1999) 204–228.
[8] G. Weerasinghe, I. Antonois and L. Lipsky, An analytic performance model of parallel systems that perform $N$ tasks using $P$ processors that can fail, in: *International Symposium on Network Computing and Applications*, Cambridge, MA (February 2002).
[9] J.H. Holland, *Adaptation in Natural and Artifical Systems* (Univ. of Michigan Press, 1975).
[10] D. Beasley, D.R. Bull and R.R. Martin, An overview of genetic algorithms: Part 1, fundamentals, University Computing 15 (February 1993) 58–69.
[11] G. Rudolph, Convergence analysis of canonical genetic algorithms, IEEE Trans. Neural Networks 5 (January 1994) 96–101.
[12] D.E. Goldberg and K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: *Foundations of Genetic Algorithms* (Morgan Kaufmann, 1991) pp. 69–93.
[13] L. Wang, H.J. Siegel, V.P. Roychowdhury and A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, Journal of Parallel and Distributed Computing 47 (November 1997) 8–22.
[14] D.W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization* (Wiley, New York, 1992).
[15] C.H. Sauer and K.M. Chandy, *Computer Systems Performance Modeling* (Prentice-Hall, Englewood Cliffs, NJ, 1981).
[16] H. Stark and J. Woods, *Probability, Random Processes, and Estimation Theory for Engineers* (Prentice-Hall, Englewood Cliffs, NJ, 1994).
[17] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen and S. Ali, Task execution time modeling for heterogeneous computing systems, in: *IPDPS Workshop on Heterogeneous Computing*, Cancún, Mexico (May 2000) pp. 185–199.
[18] P.A. Dinda, Online prediction of the running time of tasks, Cluster Computing 5 (2002) 225–236.

**Atakan Doğan** received the M.S. degree in electrical and electronics engineering from Anadolu University in 1997, and the Ph.D. degree in electrical engineering from the Ohio State University in 2001. He worked as a post doctoral researcher at the Ohio State University for nine months and joined the Department of Electrical and Electronics Engineering, Anadolu University in 2002, where he is currently an Assistant Professor. His current research interests include cluster and grid computing, interprocessor communication on parallel architectures, and wireless networks.
E-mail: atdogan@anadolu.edu.tr

**Füsun Özgüner** received the M.S. degree in electrical engineering from the Istanbul Technical University in 1972, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana-Champaign, in 1975. She worked at the IBM T.J. Watson Research Center with the Design Automation group for one year and joined the faculty at the Department of Electrical Engineering, Istanbul Technical University in 1976. Since January 1981 she has been with The Ohio State University, where she is presently a Professor of Electrical Engineering. Her current research interests are parallel and fault-tolerant architectures, heterogeneous computing, reconfiguration and communication in parallel architectures, real-time parallel computing and communication, wireless networks and parallel algorithm design. Dr. Ozgiiner has served as an Associate Editor of the IEEE Transactions on Computers and on Program Committees of several international conferences.
E-mail: ozguner@ee.eng.ohio-state.edu