Stochastic Approach to Scheduling Multiple Divisible Tasks on a Heterogeneous Distributed Computing System

Ankur Kamthe[1] and Soo-Young Lee[2]

[1]University of California, Merced
Computer Science and Engineering
Merced, CA 95344 USA
akamthe@ucmerced.edu

[2]Auburn University
Dept. of Electrical and Computer Engineering
Auburn, AL 36849 USA
leesooy@eng.auburn.edu

Abstract

Heterogeneity has been considered in scheduling, but without taking into account the temporal variation of completion times of the sub-tasks for a divisible, independent task. In this paper, the problem of scheduling multiple, divisible independent tasks on a heterogeneous distributed computing system is addressed. The "stochastic" approach, which was previously applied to DAG scheduling, is employed for scheduling a group of multiple divisible as well as whole independent tasks. It explicitly considers the standard deviations (temporal heterogeneity) in addition to the mean execution times in deriving a schedule, in order to model more closely what would actually happen "on average" on a temporally heterogeneous system (instead of approximating the random weights by their means only as in other approaches). Through an extensive computer simulation, it has been shown that the proposed approach can improve schedules significantly over those by a scheme which uses the average weights only.

1 Introduction

Divisible tasks are generally divided into smaller load fractions so that they can be assigned to multiple processors for faster execution. Various scheduling algorithms exist for directed acyclic task graphs (DAGs) where there are precedence relationships among tasks, and independent tasks where there are no precedence relationships among tasks. Generally, scheduling schemes adopt some kinds of heuristics for mapping the tasks onto processors to optimize (reduce) the overall parallel execution time. In these heuristics, the computation and communication times are considered to be *deterministic*, i.e., do not vary with time.

1-4244-0910-1/07/$20.00

These days, resources in a distributed computing system such as a cluster and a computational grid are shared by multiple users. Therefore, the computing power available for a task may vary with time, i.e., *temporal heterogeneity* [1]. Also, resources may not be all identical or a different resource may exhibit a different behavior of availability, i.e., *spatial heterogeneity*. In addition, an application itself may be inherently *random*, e.g., simulated annealing, genetic algorithm, etc. Hence, the computation and communication times are no longer deterministic on such a heterogeneous distributed computing system even when all processors in the system are identical.

For tackling the problem posed by heterogeneity in task scheduling, many approaches have been proposed by a number of researchers. In DAG scheduling, spatial heterogeneity has been considered in scheduling [2][3][4]. In [5], task computation time is treated as a random variable and is statistically estimated from past observations. In [6], the standard deviations of computation and communication times were used to model the behavior of sub-tasks in a DAG instead of using means only as in other approaches. It was shown that a significant improvement can be achieved by considering the standard deviation when heterogeneity is present.

In [7], the concepts of "machine heterogeneity" and "task heterogeneity" are employed in an effort to simulate different heterogeneous computing environments in evaluating the behavior of the task mapping heuristics. However, the (temporal) heterogeneity is not taken into account in the machine and task heterogeneity. In [8], a stochastic approach was taken to scheduling a set of *independent* tasks using a Genetic Algorithm wherein computation times are not deterministic. In [9], Schopf et al focused on load-balancing a divisible task wherein the amount of data assigned to a processor for processing depends on the mean and standard deviation of execution time. But, this strategy

does not account for the need for scheduling multiple divisible (and indivisible) tasks concurrently, wherein these tasks are reordered for execution such that the total makespan is minimized. In [10], an optimal steady-state scheduling strategy was proposed for a suite of identical, independent problems where each problem consists of a set of tasks. It attempted to exploit the mixed data and task parallelism on heterogeneous clusters and grids. In [11], a method for scheduling independent tasks consisting of nonlinear DAGs was proposed. However, heterogeneity was not considered. In [12], the problem of allocating and scheduling a collection of independent, equal-sized tasks on heterogeneous computing platforms was addressed, taking memory constraints into account. In [13], the problem of scheduling multiple divisible load applications on a grid platform was addressed. They did not explicitly consider the heterogeneity in completion times of the tasks executed on a cluster when scheduling successive tasks.

As discussed above, the existing scheduling schemes are mainly targeted towards DAGs or independent tasks but not a hybrid mix of divisible and indivisible tasks. In this paper, an effective scheme for scheduling a group of multiple divisible as well as whole independent tasks is proposed based on the stochastic approach previously designed for DAG scheduling [6]. It explicitly considers the standard deviations (temporal heterogeneity) in addition to the means of execution times in deriving a schedule. By utilizing the second order moments of weights also, the heuristics employed in the new approach attempt to "follow" more closely "on average" what would actually happen on a temporally heterogeneous system. Through an extensive computer simulation, it has been shown that the proposed approach can improve schedules significantly over those by a scheme which uses the average weights only.

The rest of this paper is organized as follows. In Section 2, the terms and notations used in the paper are introduced. In Section 3, the scheduling heuristics used in the paper are reviewed. In Section 4, the divisible load scheduling model is described. In Section 5, the new approach to scheduling groups of divisible tasks is described. In Section 6, the description of simulation procedure and performance measures is provided. In Section 7, the simulation results are discussed in detail, followed by the summary in Section 8.

2 Terms and Notations

- P_j: processor j.
- n_i: task i.
- t_{ij}: computation time of n_i, which has the mean m_{ij} and standard deviation σ_{ij} on P_j.
- t_{cij}: completion time of n_i, which has the mean m_{cij} and standard deviation σ_{cij} on P_j.
- Schedule length, T_{SL}: the make-span of a schedule, or the parallel execution time *predicted* by a scheduling algorithm.
- Average parallel execution time, t_p: the actual average time required to execute a group of tasks according to a schedule. Parallel execution time, T_p, which is a random variable has the mean, t_p, and the standard deviation, σ_{T_p}.
- E[ ]: the expectation operator used to compute the mean of a random variable.

3 Scheduling Heuristics

Two existing heuristics developed for scheduling independent tasks, i.e., Max-Min and Min-Min, which are used to demonstrate the benefits of the proposed approach, are reviewed below. They have time complexities of $O(V^2P)$, where P is the number of processors and V is the number of tasks to schedule. These simple heuristics were chosen as the main goal of this study was to demonstrate that significant improvements in t_p could be achieved only by improving estimation of time variables (such as earliest start time, completion time) while scheduling tasks. The proposed idea would also apply to any other scheduling algorithm utilizing time variables in computing the makespan.

Min-Min Algorithm

1. A task list is generated that includes all unmapped tasks.
2. Find the completion time (CT) of each unmapped task on each machine (ignoring other unmapped tasks).
3. Find the machine that gives minimum CT for each task.
4. Among all the task/machine pairs found in 3, find the pair that gives the minimum CT.
5. Remove the above task from the task list and map it to the chosen machine.
6. Update the available time of the machine on which the task is mapped.
7. Repeat steps 2-6 until all the tasks have been mapped.

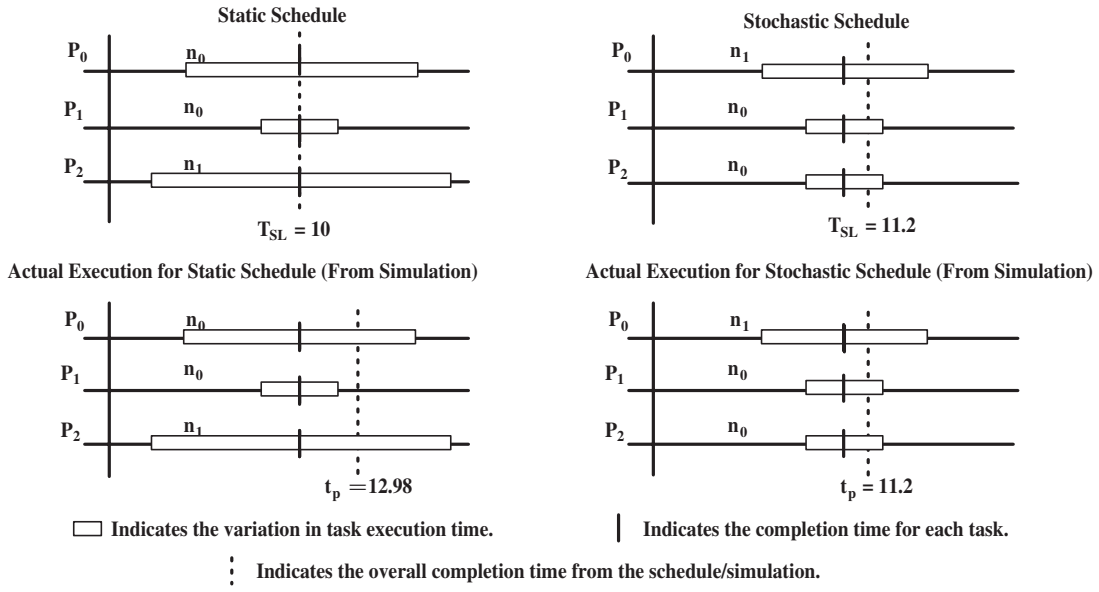

Figure 1. Comparison of schedule and average parallel execution time from the static and stochastic schedules.

Table 1. The mean and standard deviations of task execution times used for the comparison in Figure 1. Task n_0 is divided into two sub-tasks, and m and σ given in the table are for each of the sub-task.

Task	m_{ci0}, σ_{ci0}	m_{ci1}, σ_{ci1}	m_{ci2}, σ_{ci2}
n_0	$(10, 3)$	$(10, 1)$	$(10, 1)$
n_1	$(10, 2)$	$(10, 4)$	$(10, 4)$

Max-Min Algorithm

The Max-min algorithm is the same as the Min-min algorithm except that in Step 4 "find the pair that gives the minimum CT" is replaced by "find the pair that gives the maximum CT."

4 Divisible Task Scheduling

One essential step in both Max-Min and Min-Min schemes is to compute the CT of each task. When a group of independent tasks is executed on a temporally (and spatially) heterogeneous distributed computing system, the CT of each task may vary with time in general. That is, CT of n_i can be considered to be a random variable (t_{cij}). Hence, there is an uncertainty in the CT of the task due to the heterogeneity in resources. Consider the following cases:

- Case 1: When an independent task requiring multiple processors is executed on a temporally heterogeneous but spatially homogeneous system, t_{cij} would change with respect to time in a random manner.
- Case 2: When an independent task requiring multiple processors is executed on a spatially heterogeneous but temporally homogeneous system, the processor-dependent CT of a task can be modeled by the random variables t_{cij}.
- Case 3: When an independent task requiring multiple processors is executed on a temporally and spatially heterogeneous system, the random variables t_{cij} reflects both temporal and spatial variation of CT of n_i.

In reality, heterogeneity is comprised of a combination of temporal and spatial heterogeneity, and can be characterized as one of Cases 1-3 mentioned above or a mix of these three cases where task heterogeneity is combined with spatial or temporal heterogeneity. In the remainder of this paper, the first case is assumed for the convenience of discussion though all three cases can be handled by the stochastic model. In our model for divisible task scheduling, the ran-

dom variable t_{cij} is replaced by a pair of its first two moments, i.e., the mean m_{cij} and standard deviation σ_{cij}. Similarly, the random variable t_{cik} is replaced by its mean m_{cik} and standard deviation σ_{cik}. In order to justify the proposed approach, consider the following examples. It is assumed that when a divisible task is partitioned into p sub-tasks, it can be scheduled only if at least p processors become available.

Case A: Difference in computing completion time for a divisible task.
Suppose two independent tasks are to be scheduled, one of the tasks (n_0) is divisible and requires two processors for execution (refer to Table 1 for task execution times and Figure 1). In the static case (refer to Figure 1), the CT is computed to be $t_{cij} = m_{cij}$ for all tasks (hence, n_0 has the same CT (= 10) on all processors). Therefore, n_0 is scheduled on P_0 and P_1, and n_1 is scheduled on P_2, leading to schedule length $T_{SL} = 10$. Execution of this schedule leads to $t_p = 12.98$. In the stochastic case, the CT of n_0 is computed as $E[max(t_{c0j}, t_{c0k})]$ and hence is different for each pair of P_j and P_k ($j \neq k$) (least CT (= 10.9) if scheduled on P_1 and P_2). CT of n_1 (t_{c1j}) remains the same for all j. Therefore, n_0 is scheduled on P_1 and P_2, and n_1 is scheduled on P_0. The schedule length is computed as $E[max(t_{c10}, t_{c01}, t_{c02})]$ in the stochastic case. This leads to $T_{SL} = 11.2$. Execution of this schedule leads to $t_p = 11.2$, which is the same as T_{SL} predicted before.

Case B: Difference in computing completion time for a divisible task scheduled after independent tasks.
Suppose that four independent tasks are to be scheduled, where one of the tasks (n_3) is divisible and requires two processors for execution (refer to Table 2 for task execution times and Figure 2). In the static case, tasks n_0, n_1 and n_2 are scheduled on P_0, P_1 and P_2, respectively. n_3 which is a divisible task is scheduled on P_0 and P_1 (for the static case, CT for a divisible task is the same for all processor pairs (= 25)). This leads to $T_{SL} = 25$. During execution of this schedule, the start of n_3 is delayed due to the heterogeneity in execution times of n_0 and n_1. This leads to $t_p = 31.05$. In the stochastic case, just as in the static case, tasks n_0, n_1 and n_2 are scheduled on P_0, P_1 and P_2 respectively. The CT of n_3 is computed as $E[max(t_{c00}, t_{c11})] + E[max(t_{30}, t_{31})]$ (or $E[max(t_{c11}, t_{c22})] + E[max(t_{31}, t_{32})]$ or $E[max(t_{c00}, t_{c22})] + E[max(t_{30}, t_{32})]$). It is least ($\approx$ 28) when n_3 is scheduled on P_1 and P_2. Thus, the stochastic case considers the heterogeneity in the start time for a divisible task and uses it while computing the CT for that task. This leads to $T_{SL} = 28.1$ (computed as $E[max(t_{c00}, t_{c31}, t_{c32})]$). Execution of this schedule leads to $t_p = 28.1$, which is the same as T_{SL} predicted before.

Parallel execution time of a group of independent tasks is defined to be the total time required to execute all the tasks. Since the parallel execution time (T_p) is random, one may adopt the *average parallel execution time* $t_p = E[T_p]$ to quantify the quality of a schedule. As mentioned in Section 1, this measure is useful especially when a task needs to be repeatedly executed.

5 Stochastic Approach to Divisible Task Scheduling

In this section, a new approach to scheduling divisible tasks is described. Its implementations for Min-Min and Max-Min are referred to as *stochastic* Min-Min and *stochastic* Max-Min, respectively. It is assumed that $\{m_{ij}, \sigma_{ij}\}$ is known at the time of scheduling, i.e., static scheduling.

5.1 Completion Time for a divisible independent task

In the stochastic Min-Min (or Max-Min), the *average* CT (just CT hereafter) of each node is used in computing the priority of the node for scheduling. One of the differences between the stochastic and static versions of Min-Min and Max-Min is how the CT of a divisible independent task is computed when it is executed concurrently on multiple processors. Referring to Figure 1, the CT of n_0 is $E[max(t_{c01}, t_{c02})]$. When $m_{c01} \gg m_{c02}$ (or $m_{c01} \ll m_{c02}$), then $E[max(t_{c01}, t_{c02})] \approx m_{c01}$ (or $E[max(t_{c01}, t_{c02})] \approx m_{c02}$). However, when m_{c01} is comparable to m_{c02} (this is mostly the case due to load balancing), and σ_{c01} or $\sigma_{c02} \neq 0$, $E[max(t_{c01}, t_{c02})]$ is substantially greater than $max(m_{c01}, m_{c02})$. In general, the CT for a divisible task i requiring p processors for execution can be expressed as $E[max(t_{ci1}, ..., t_{cip})]$. Referring to Figure 2, the CT of n_3 is $E[max(t_{c11}, t_{c22})] + E[max(t_{31}, t_{32})]$. However, when m_{c11} is comparable to m_{c22}, m_{31} is comparable to m_{32} and $\sigma_{c11}, \sigma_{c22}, \sigma_{31}$ or $\sigma_{32} \neq 0$, $E[max(t_{c11}, t_{c22})] + E[max(t_{31}, t_{32})]$ is substantially greater than $max(t_{c11}, t_{c22}) + max(t_{31}, t_{32})$.

These cases are referred to as *competing situations*, i.e., multiple tasks compete to determine the completion time of n_0 (or n_3). The competing situation implies that the divisible portions of n_0 (or n_3) are scheduled on multiple different processors. Note that a node may not be scheduled on the processors involved in executing the divisible portions of a task, until after all of the sub-tasks of a divisible task complete execution or until the independent tasks executing on processors, where the divisible task would be scheduled, have completed execution.

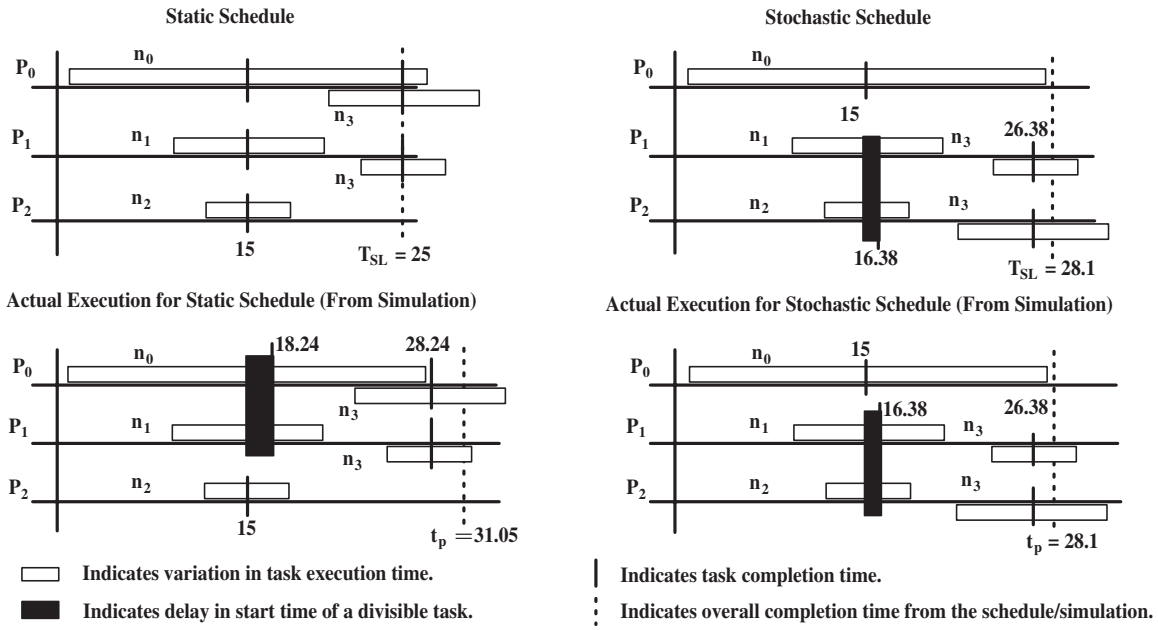

Figure 2. Comparison of schedule and average parallel execution time from the static and stochastic schedules.

Table 2. The mean and standard deviations of task execution times used for the comparison in Figure 2. Task n_3 is divided into two sub-tasks, and m and σ given in the table are for each of the sub-tasks.

Task	m_{ci0}, σ_{ci0}	m_{ci1}, σ_{ci1}	m_{ci2}, σ_{ci2}
n_0	$(15, 7)$	$(15, 4)$	$(15, 6)$
n_1	$(15, 4)$	$(15, 3)$	$(15, 4)$
n_2	$(15, 3)$	$(15, 2)$	$(15, 1)$
n_3	$(10, 3)$	$(10, 1)$	$(10, 3)$

5.2 Derivation of CT

Calculation of the CT of a divisible task n_i requires evaluation of $E[max\{S_i\}]$ where S_i is the set of random variables which determine the CT of the task. For example, in Figure 1, $S_0 = \{t_{c01}, t_{c02}\}$ for n_0 if it is scheduled on P_1 and P_2 (t_{cij} indicates the time for each divisible portion of n_i executed on P_j). One may derive an analytic formula of $E[max\{S_i\}]$ so that it can be used during scheduling. However, it is not possible to derive the closed-form analytic formula of $E[max\{S_i\}]$ for any distribution of each random variable in S_i. In such a case, one may approximate $E[max\{S_i\}]$ in terms of the mean and standard deviation of each random variable in S_i. In this study, the closed-form analytic formulas of $E[max\{S_i\}]$ have been derived for the uniform distribution of each random variable in S_i.

5.3 Complexity

The complexity of the stochastic Max-Min is the same as that of the static Max-Min, which is $\boldsymbol{O}(V^2P)$. However, the step of deriving the CT of each node requires more computation in the stochastic Min-Min and Max-Min which use the formulas mentioned in Section 5.2, compared to the static Min-Min and Max-Min. In Figure 3, the average computation time required for deriving a schedule is compared, as a function of the number of processors, among the different scheduling schemes. It is seen that the stochastic approaches at times took approximately two times longer than the static approaches. However, it is still less than 3 seconds in all the cases considered. Also, note that the scheduling overhead is incurred just once.

Table 3. Comparison of schedule length (T_{SL}) and t_p and improvements in t_p by the stochastic Max-Min over the static Max-Min when the competing situation is significant in the presence of temporal heterogeneity in task execution times.

number of tasks	number of processors	Static Max-Min					Stochastic Max-Min						
		T_{SL}	t_p	σ_{T_p}	Δ	δ	T_{SL}	t_p	σ_{T_p}	Δ	δ	η	$\sharp count$
100	2	686.4	758.3	21.7	71.9	10.0	738.4	721.6	25.1	16.8	2.3	4.8	20
	3	455.8	525.7	20.6	69.9	14.2	518.7	515.6	19.3	3.1	0.6	1.9	160
	4	343.1	397.8	17.1	54.7	14.8	390	378.5	14.9	11.5	3.0	4.9	62
	5	276.9	326.1	15	49.2	16.3	307.9	309.9	12.7	2	0.6	5	100
	6	227.5	271.7	12.1	44.2	17.7	259.7	260.5	11.4	0.8	0.3	4.1	113
	7	193.3	234.8	9.7	41.5	19.4	222.8	230.1	10.6	7.3	3.2	2	185
	8	169.5	207.1	8.6	37.6	20.0	195	199.7	9.1	4.7	2.4	3.6	139
200	2	1281.8	1437.3	32.5	155.5	11.4	1357.4	1373.9	32.2	16.5	1.2	4.4	10
	3	855.4	979.5	24.1	124.1	13.5	985.3	975.8	24.5	9.5	1.0	0.4	231
	4	641.8	750.5	20.2	108.7	15.6	721.5	707.1	20.2	14.4	2.0	5.8	22
	5	514	611.1	18.3	97.1	17.3	592	588.5	17.7	3.5	0.6	3.7	75
	6	427.8	509.2	14.4	81.4	17.4	489.8	490.9	15.5	1.1	0.2	3.6	77
	7	367.3	438	13.5	70.7	17.6	414.5	425.1	13.5	10.6	2.5	3	132
	8	321.4	387.5	12.2	66.1	18.6	370.2	374.9	12.1	4.7	1.3	3.2	110
500	2	3316.9	3653.1	52.8	336.2	9.6	3479.6	3457.8	51.6	21.8	0.6	5.3	0
	3	2216.3	2490.6	37.2	274.3	11.7	2450	2429.9	41.6	20.1	0.8	2.4	50
	4	1654.8	1886.6	30.4	231.8	13.1	1794.9	1773.1	33.3	21.8	1.2	6	0
	5	1320.7	1537.4	28.7	216.7	15.2	1508.7	1492.3	30	16.4	1.1	2.9	73
	6	1099.2	1278.7	24.3	179.5	15.1	1233.6	1205.5	25.2	28.1	2.3	5.7	6
	7	940.8	1100.6	21.1	159.8	15.7	1088.4	1081.8	24.4	6.6	0.6	1.7	141
	8	826.4	973.7	19	147.3	16.4	948.1	927.4	21.1	20.7	2.2	4.8	25

Table 4. Comparison of schedule length (T_{SL}) and t_p and improvements in t_p by the stochastic Min-Min over the static Min-Min when the competing situation is significant in the presence of temporal heterogeneity in task execution times.

number of tasks	number of processors	Static Min-Min					Stochastic Min-Min						
		T_{SL}	t_p	σ_{T_p}	Δ	δ	T_{SL}	t_p	σ_{T_p}	Δ	δ	η	$\sharp count$
100	2	1339	1526.6	29.5	187.6	13.1	1552.1	1514.7	28.9	37.4	2.4	0.8	87
	3	1281	1474	29.1	193	14.0	1419.9	1433.9	23.2	14	1.0	2.7	39
	4	676.8	791.1	18	114.3	15.6	746.8	767.3	13.8	20.5	2.7	3	61
	5	648	774.2	18.3	126.2	17.7	705.1	725.7	12.7	20.6	2.9	6.3	6
	6	454.3	534.9	13.8	80.6	16.3	542.4	522.5	10.6	19.9	3.7	2.3	105
	7	434.8	529.6	13.7	94.8	19.7	480	493.5	10.1	13.5	2.8	6.8	7
	8	345	412	10.6	67	17.7	406	396.2	8.4	9.8	2.4	3.8	53
200	2	2615	3006.2	40.2	391.2	13.9	3062.4	2995.6	41.1	66.8	2.2	0.4	59
	3	2503.6	2892.2	41.2	388.6	14.4	2875.9	2841.5	34.5	34.4	1.2	1.8	50
	4	1310.8	1518.6	24.1	207.8	14.7	1484.5	1476.9	20.3	7.6	0.5	2.7	30
	5	1257	1497.5	25.3	240.5	17.5	1388.4	1401	15.6	12.6	0.9	6.4	0
	6	878.9	1026	18.6	147.1	15.4	1020.4	995.9	12.7	24.5	2.4	2.9	44
	7	842.6	1026	18.3	183.4	19.6	929.4	945.4	10.9	16	1.7	7.9	0
	8	662.3	789.2	17.4	126.9	17.5	773.9	753	9.1	20.9	2.7	4.6	11
500	2	6380.8	7286.4	68	905.6	13.3	7315.6	7270.5	67.5	45.1	0.6	0.2	44
	3	6050.9	6949.4	61.6	898.5	13.8	6829.6	6887.4	52.5	57.8	0.8	0.9	86
	4	3197.1	3720.2	40.2	523.1	15.1	3515.8	3592	30.1	76.2	2.1	3.4	2
	5	3031.9	3599.6	36.7	567.7	17.1	3479.2	3419.6	24.1	59.6	1.7	5	0
	6	2136.2	2496.4	30.8	360.2	15.6	2364.5	2397	19.8	32.5	1.4	4	1
	7	2015	2441.9	26.4	426.9	19.2	2211.5	2264.9	16.8	53.4	2.4	7.2	0
	8	1605.6	1892	22.8	286.4	16.4	1850.5	1805.4	13.9	45.1	2.5	4.6	0

Table 5. Comparison of schedule length (T_{SL}) and t_p and improvements in t_p by the stochastic Min-Min over the static Min-Min when the competing situation is significant in presence of temporal and spatial heterogeneity in task execution times.

number of tasks	number of processors	Static Min-Min					Stochastic Min-Min						
		T_{SL}	t_p	σ_{T_p}	Δ	δ	T_{SL}	t_p	σ_{T_p}	Δ	δ	η	$\sharp count$
40	2	703.9	778.1	21	74.1	10.0	795.5	778.1	21	17.4	2.2	0	500
	3	649.4	754.2	23.1	104.8	14.9	768.2	745.5	18	22.7	3.0	1.2	110
	4	353	400.7	12.8	47.7	12.7	403.1	394.3	11	8.9	2.2	1.6	19
	5	333.9	389.4	12.7	55.4	15.3	368.2	380.2	10.1	11.9	3.2	2.4	165
	6	232	273.3	9.5	41.4	16.4	290.6	273.1	8.5	17.5	6.2	0.1	124
	7	228.8	269.7	9.9	40.9	16.4	252.9	261	8.9	8.1	3.2	3.2	42
	8	184.3	217.8	9.9	33.6	16.7	216.8	204.5	5.5	12.3	5.8	6.1	85
200	2	2900.5	3221.5	44.3	320.9	10.5	3239.5	3210	42.1	29.5	0.9	0.4	141
	3	2587.5	3022.7	43.8	435.2	15.5	2925	3018.4	35.8	93.3	3.1	0.1	213
	4	1431.1	1628.8	28.1	197.8	12.9	1648.7	1604.3	21.2	44.4	2.7	1.5	99
	5	1280.3	1544.1	25.7	263.8	18.7	1540.5	1505.2	19.6	35.3	2.3	2.5	43
	6	942.3	1104.5	20.4	162.2	15.8	1054	1068.5	15.5	14.5	1.4	3.3	36
	7	856.4	1052.3	20.3	195.9	20.5	979.5	1012.3	13.2	32.9	3.3	3.8	20
	8	704.5	845.7	16.9	141.3	18.2	820.6	804.4	12.2	16.2	2.0	4.9	9
500	2	7496	8344	67.4	848	10.7	8396.3	8338	67.7	58.3	0.7	0.1	186
	3	6759.2	7860.5	64	1101.3	15.1	7854.7	7885.7	54.1	31	0.4	-0.3	334
	4	3657.4	4179.7	43.4	522.3	13.3	4007.7	4083.5	31.9	75.8	1.9	2.3	12
	5	3353.1	3987.5	40.5	634.4	17.3	3823.8	3896.6	26.3	72.8	1.9	2.3	10
	6	2416.1	2811.7	32.4	395.6	15.1	2831.1	2716.2	20.7	114.9	4.1	3.4	1
	7	2233.6	2689.9	28.9	456.3	18.5	2528	2586.6	18	58.5	2.3	3.8	0
	8	1798.2	2125	24.7	326.8	16.7	2053.9	2043.2	16.7	10.8	0.5	3.9	2

Table 6. Comparison of schedule length (T_{SL}) and t_p and improvements in t_p by the stochastic Max-Min over the static Max-Min when the competing situation is significant in presence of temporal and spatial heterogeneity in task execution times.

number of tasks	number of processors	Static Max-Min					Stochastic Max-Min						
		T_{SL}	t_p	σ_{T_p}	Δ	δ	T_{SL}	t_p	σ_{T_p}	Δ	δ	η	$\sharp count$
40	2	413.5	464.6	21.4	51.1	11.6	460.3	453	21.8	7.3	1.6	2.5	168
	3	282.9	328.2	15.5	45.3	14.8	323	329.6	15.4	6.7	2.1	-0.4	439
	4	207.5	243.8	11.7	36.3	16.1	237.4	238.6	10.5	1.2	0.5	2.1	196
	5	166.4	201.5	9.8	35.1	19.1	196.8	194.9	10.2	1.9	1.0	3.2	320
	6	135.5	167	8.8	31.5	20.8	163.6	161.1	9.3	2.5	1.5	3.5	214
	7	120	147.1	8.4	27.1	20.3	145.2	142.3	7.2	2.9	2.0	3.3	233
	8	102.3	128.5	7.6	26.2	22.7	120.8	127.5	7.2	6.7	5.4	0.8	264
200	2	1441.1	1562.2	38.6	121.1	8.1	1539.1	1509.1	38.8	29.9	2.0	3.4	38
	3	951.9	1077.2	30.5	125.2	12.3	1050.8	1064.5	30.7	13.7	1.3	1.2	184
	4	711.6	809.1	20.8	97.5	12.8	787.6	800.6	24.8	12.9	1.6	1	186
	5	573	673.4	22.4	100.4	16.1	639.7	643.4	19.4	3.7	0.6	4.5	80
	6	477.6	563.7	16.9	86.1	16.5	548	540.2	19.2	7.9	1.5	4.2	81
	7	404.5	476.7	15.9	72.2	16.4	452.4	466.3	16	13.9	3.0	2.2	143
	8	352.7	418.8	14.2	66.1	17.1	387.9	409.5	14.9	21.6	5.4	2.2	153
500	2	6377.1	7063.4	88.5	686.3	10.2	6895.9	6845	88.1	50.9	0.7	3.1	0
	3	4306	4952.3	68.1	646.3	14.0	4919.7	4902.9	64.1	16.8	0.3	1	118
	4	3240.5	3782.8	54.9	542.2	15.4	3660.4	3676.7	51.9	16.3	0.4	2.8	22
	5	2585.8	3072.3	47.2	486.4	17.2	3019.2	3011.6	43.1	7.6	0.3	2	65
	6	2141.6	2551.1	41.7	409.5	17.5	2474.7	2493.9	39.7	19.2	0.8	2.2	74
	7	1823.5	2214.2	39.3	390.8	19.4	2140.8	2155.5	36.9	14.7	0.7	2.7	62
	8	1583.8	1933.7	33.5	349.9	19.9	1860.9	1885.6	32.6	24.7	1.3	2.5	57

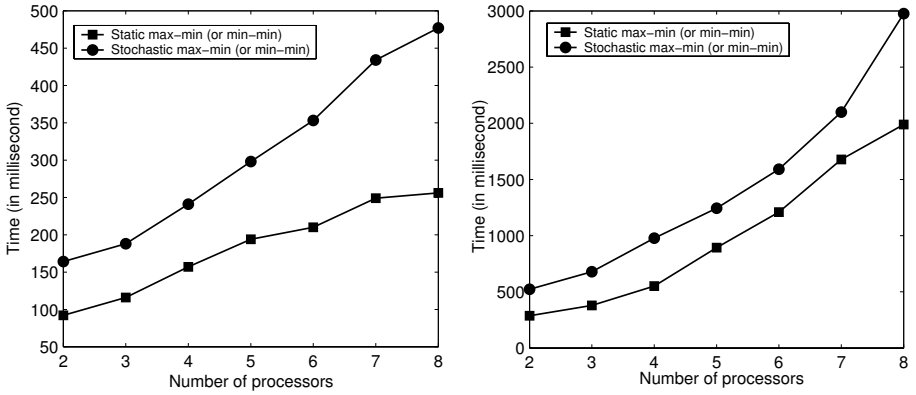

Figure 3. Average time (in milliseconds) required to derive the schedule on Sun Enterprise 420R server (4 450-MHz UltraSPARC[tm]-II modules with 4GB memory) for (a) 100 nodes and (b) 500 nodes.

6 Simulation

The simulation results are divided into two sets. In the first set (refer to Tables 3 and 4), task execution times exhibit varying temporal heterogeneity, i.e., they have the same mean but different standard deviations for different processors. In the second set (refer to Tables 5 and 6), task execution times exhibit varying spatial and temporal heterogeneity, i.e., they have different means and standard deviations for different processors (means of execution times for a task vary up to a maximum of 10% for different processors). The ranges of the means and standard deviations of execution times are given below.

- For tasks exhibiting temporal heterogeneity:
  - Divisible tasks: $m_{ij} = [20, 40]$ and $\sigma_{ij} = [0, 15]$
  - Non-divisible tasks: $m_{ij} = [0, 20]$ and $\sigma_{ij} = [0, 10]$
- For tasks exhibiting temporal and spatial heterogeneity:
  - Divisible tasks: $m_{ij} = [25, 45]$ and $\sigma_{ij} = [0, 15]$
  - Non-divisible tasks: $m_{ij} = [0, 20]$ and $\sigma_{ij} = [0, 10]$
- For Max-Min scheme, 15% of the tasks and for Min-Min scheme, 80% of the tasks are divisible whereas the rest are non-divisible.

6.1 Simulation Procedure

In order to analyze effectiveness of the new scheduling schemes (stochastic Min-Min and Max-Min), an extensive simulation has been carried out where they are compared with the static Min-Min and Max-Min in terms of the average parallel execution time t_p. The simulation procedures are described below.

1. Designate the number of tasks to be scheduled in the simulation.
2. Assign each task with a positive mean and standard deviation of execution time, which define a uniformly-distributed random variable.
3. Find a schedule using each of the static Min-Min, stochastic Min-Min, static Max-Min and stochastic Max-Min.
4. Generate random weights for all tasks according to their means and standard deviations.
5. Compute parallel execution time (T_p) for each of the schedules obtained in Step 3 using the static Min-Min, stochastic Min-Min, static Max-Min and stochastic Max-Min algorithms.
6. Repeat Steps 4 and 5 multiple times to obtain the average parallel execution time (t_p).

Step 6 is repeated 500 times in order to obtain simulation results (t_p).

6.2 Performance Measures

In addition to t_p, the following measures are used:

- Δ: the difference between the schedule length, T_{SL}, and the average parallel execution time, t_p, which quantifies the accuracy of a scheduling scheme, i.e.,

$$\Delta = |T_{SL} - t_p|.$$

- δ: the percent difference between T_{SL} and t_p, normalized by the average of T_{SL} and t_p, i.e.,

$$\delta = \frac{\Delta}{\frac{T_{SL}+t_p}{2}} \times 100 = \frac{|T_{SL} - t_p|}{\frac{T_{SL}+t_p}{2}} \times 100$$

- η: the percent improvement in t_p by the stochastic algorithms (Min-Min and Max-Min) over their respective static versions, i.e.,

$$\eta = \frac{t_p^{static} - t_p^{stochastic}}{t_p^{static}} \times 100$$

where t_p^{static} and $t_p^{stochastic}$ are the average parallel execution times achieved by the static and stochastic Min-Min (or Max-Min), respectively.

- $\sharp count$: the number of times (runs) during each simulation (500 runs) that T_p for the stochastic algorithms (Min-Min and Max-Min) is greater than the corresponding T_p from their respective static versions,i.e., a lower $\sharp count$ is an indication that the stochastic algorithm outperformed its static counterpart more consistently.

7 Results and Discussion

7.1 Effects of Temporal Heterogeneity

In Tables 3, 4, 5 and 6, the schedule length (make-span), T_{SL}, and the average parallel execution time, t_p, achieved by the schedule are compared. It can be seen that T_{SL} and t_p for the stochastic Min-Min and Max-Min match very closely, i.e., small δ. This small difference (δ) is due to the accurate estimation of CT in a temporally heterogeneous environment by considering the standard deviation (σ_i) in addition to the mean (m_i) of computation time for each task (n_i) (divisible and otherwise). However, there is a significant difference between T_{SL} and t_p in the static Min-Min and Max-Min which use only m_i ignoring σ_i, i.e., temporal heterogeneity. This is because effect of temporal heterogeneity on the CT of a task in the competing situation (due to presence of divisible tasks) becomes significant, but the static Min-Min and Max-Min ignore it. They tend to "under-estimate" the CT of tasks (n_k) involved in the competing situation since $E[max((t_{cij}), (t_{cik}))] > max((m_{cij}), (m_{cik}))$. Therefore, it often ends up with a premature scheduling of further tasks on processors which may be executing separate portions of a divisible task, leading to a worse schedule resulting in a longer t_p. Also, the under-estimation is a source for the large discrepancy between T_{SL} and t_p.

7.2 Improvements by Stochastic Min-Min and Max-Min

In Tables 3, 4, 5 and 6, the stochastic Max-Min and Min-Min are compared to the static Max-Min and Min-Min also in terms of *percent improvement*, η.

When the competing situation is present, a large improvement has been achieved by the stochastic Max-Min and Min-Min (refer to Tables 3, 4, 5 and 6). This significant improvement is due to the accurately estimated CT for divisible, independent tasks and its proper use in scheduling. Also, it can be observed that there is only a very small difference in variation of T_p between the static Max-Min (or Min-Min) and stochastic Max-Min (or Min-Min). It is noticed that in each of the simulations the improvement varies significantly with the number of processors used. The "degree" of competition in a competing situation depends not only on the number of divisible tasks and execution times but also on the number of processors. Suppose that a task is divided into two sub-tasks where their average computation times are the same or similar and at least one of them shows significant temporal heterogeneity. When these subtasks are scheduled concurrently, it will lead to a competing situation and any further tasks will be executed only after execution of the current divisible task is completed. A different number of processors implies a different "degree" of the competing situation. Hence, the improvement depends on the number of processors employed in scheduling. The higher the number of sub-tasks for a divisible task, the greater the "degree" of competing situation is.

One observation is that the stochastic Max-Min tends to achieve larger improvements than the stochastic Min-Min. The Min-Min schedules small tasks first and therefore most tasks (which are small) are not divided in the early stage of scheduling. That is, the competing situation in the early stage of schedule has only a small effect on t_p. The effect of a non-optimal schedule in the early stage may grow with execution of tasks and become larger in the later stage. As scheduling progresses, more tasks are partitioned. However, the Min-Min tends to end up with an unbalanced workload distribution among processors in general, hiding the competing situation in the later stage. Consequently, there is a relatively smaller improvement by the stochastic Min-Min over the static Min-Min. On the other hand, the Max-Min schedules larger tasks first and usually finds a schedule where the workload distribution among processors is well-balanced. Hence, the improvement by the stochastic Max-Min over the static Min-Min is larger than that by the stochastic Min-Min over the static Min-Min.

It is also observed in the tables that η is larger for the cases with temporal heterogeneity only than for those with both temporal and spatial heterogeneity. When different processors have different effective speeds (availability), it

Table 7. Variation in η with number of tasks.

Scheduling Scheme	number of tasks	η			
		Average	Maximum	Minimum	p-value
Max-Min	40	1.99	6.12	-3.13	< 0.001
	100	2.71	5.95	-1.52	< 0.001
	200	3.21	7.48	-1.56	< 0.001
	500	3.56	6.71	-0.26	< 0.001
Min-Min	40	2.45	7.5	-4.05	< 0.001
	100	2.86	8.51	-2.23	< 0.001
	200	2.91	8.15	-1.18	< 0.001
	500	3.21	8.39	-0.14	< 0.001

is less likely to get a significant level of competing situation and, therefore, a smaller improvement by the stochastic schemes is achieved. Also, it is partially due to the way a divisible task is partitioned in the simulation, i.e., uniform partitioning.

7.3 Dependency on number of tasks

In Table 7, dependency of η on the number of tasks is analyzed (the number of processors ranges from 2-8). Here, it is seen that as the number of tasks involved in the simulation increases, η increases for both Max-Min and Min-Min scheduling schemes. A greater number of tasks makes effects of the competing situation more pronounced leading to a larger η achieved by considering the effects in scheduling. The last column in the table indicates the one-sided p-value from the Wilcoxon Signed Rank test (95% confidence level). For all cases, the p-value is < 0.001 indicating that the results are statistically significant, i.e., a reduction in t_p is achieved by using the stochastic scheduling approach over the static approach.

8 Summary

A conventional scheduling scheme which considers only the means of task execution times is not able to find the best possible schedule in a heterogeneous environment. In this paper, a new approach to scheduling a group of independent divisible and non-divisible tasks is proposed and its performance has been analyzed through simulation. The first implementations of the approach based on the Max-Min and Min-Min algorithms, called the stochastic Max-Min and Min-Min, have well demonstrated that the schedules derived by the proposed approach are significantly better in terms of the average parallel execution time than those by the static Max-Min and Min-Min which consider only the average execution times of tasks. Also, the stochastic Max-Min and Min-Min are able to accurately predict the actual performance one can expect on a temporally heterogeneous distributed computing system, i.e., the schedule length obtained by the stochastic Max-Min and Min-Min is very close to the average parallel execution time. While the Max-Min and Min-Min were considered in this study, it should be clear that the proposed approach is applicable to any other scheduling schemes which use time parameters in computing the priority levels of tasks.

References

[1] S.-Y. Lee and J. Huang, "A Theoretical Approach to Load Balancing of a Target Task in a Temporally and Spatially Heterogeneous Grid Computing Environment," *GRID 2002*, pp. 70-81.

[2] Z. Liu, B. Fang, Y. Zhang and J. Tang, "Scheduling algorithms for a fork DAG in a NOWs," *the Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, Vol. 2* , pp. 959-960, May 2000.

[3] H. Topcuoglu, S. Hariri, and M. Wu, " Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, March 2002.

[4] R. Bajaj and D. P. Agarwal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Transactions on Parallel and Distributed Systems, Vol. 15 No. 2*,pp. 107-118 February 2004.

[5] M. A. Iverson, F. Ozguner and Lee C. Potter, "Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment," *the 8th Heterogeneous Computing Workshop (HCW '99)*, p. 99, April 1999.

[6] A. Kamthe and S.-Y. Lee, "A Stochastic Approach to Estimating Earliest Start Times of Nodes for Scheduling DAGs on Heterogeneous Distributed Computing Systems," *13th Heterogeneous Computing Workshop*, April 2005.

[7] S. Ali, H. Siegel, M. Maheswaran, D. Hensgen and S. Ali, "Task Execution Time Modeling for Heterogeneous Computing Systems," *IEEE 9th Heterogeneous Computing Workshop*, pp185-199, May 2000.

[8] A. Dogan and F. Ozguner, "Stochastic scheduling of a meta-task in heterogeneous distributed computing," *International Conference on Parallel Processing Workshops*,pp. 369-374, 2001.

[9] J.M. Schopf and F. Berman, "Stochastic scheduling," *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, November 1999.

[10] O. Beaumont, A. Legrand and Y. Robert, "Scheduling strategies for mixed data and task parallelism

on heterogeneous clusters and Grids," *Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing*, Feb. 2003.

[11] Q. Hug, Z.-G. Chen and F.C.M. Lau, "A new method for independent task scheduling in nonlinearly DAG clustering," *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks*, 2004.

[12] O. Beaumont, A. Legrand, L. Marchal and Y. Robert, "Independent and divisible tasks scheduling on heterogeneous star-shaped platforms with limited memory," *13th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, Feb. 2005.

[13] L. Marchal, Y. Yang, H. Casanova, and Y. Robert, "Steady State Scheduling of Multiple Divisible Load Applications on Wide-Area Distributed Computing Platform," *to appear in the International Journal of High Performance Computing Applications.*

Biographies

Ankur Kamthe received his Bachelors degree in Electronics from Mumbai University, India in 2002. He received his Master of Science degree in Electrical and Computer Engineering at Auburn University, Alabama in August 2005 and a Masters degree in Probability and Statistics from Auburn in August 2006. He is currently pursuing his Ph.D. degree in Computer Science and Engineering at the University of California, Merced. His research interests are heterogeneous computer networks, parallel and distributed systems programming, systems research in wireless sensor networks and statistical properties of radio links.

Soo-Young Lee received his B.S. degree in Electronics Engineering from Seoul National University, Korea, his M.S. degree in Electrical and Electronic Engineering from Korea Advanced Institute of Science, and his Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin. He was an instructor in Department of Electronics Engineering at Kyung-Pook National University, Korea, an assistant professor in School of Electrical Engineering, Cornell University, and is currently a professor in Department of Electrical and Computer Engineering, Auburn University, Auburn, Alabama. He has extensive research experience in the areas of parallel and distributed computing (parallelization of large-scale applications, load balancing and scheduling, heterogeneous computing, optimization of high performance communication, parallel image processing), medical image reconstruction (computerized tomography), proximity effect correction in electron-beam lithography, nanofabrication, etc. Recent research activities include cluster computing for large-scale applications, exploitation of heterogeneity in distributed computing and communication systems, sensor networks and their applications, and fabrication of 3-D nanostructures using electron beam lithography.