Divisible Load Scheduling with Result Collection on Heterogeneous Systems

Abhay Ghatpande, Hidenori Nakazato, Hiroshi Watanabe
School of Global Information and Telecommunication Studies
Waseda University, Tokyo
Email: abhay@toki.waseda.jp

Olivier Beaumont
INRIA Futurs – LaBRI
Bordeaux, France
Email: olivier.beaumont@labri.fr

Abstract

Divisible Load Theory (DLT) is an established mathematical framework to study Divisible Load Scheduling (DLS). However, traditional DLT does not comprehensively deal with the scheduling of results back to source (i.e., result collection*) on heterogeneous systems. In this paper, the* DLSRCHETS *(DLS with Result Collection on HETerogeneous Systems) problem is addressed. The few papers to date that have dealt with* DLSRCHETS*, proposed simplistic* LIFO *(Last In, First Out) and* FIFO *(First In, First Out) type of schedules as solutions to* DLSRCHETS*. In this paper, a new heuristic algorithm,* ITERLP*, is proposed as a solution to the* DLSRCHETS *problem. With the help of simulations, it is proved that the performance of* ITERLP *is significantly better than existing algorithms.*

1. Introduction

Divisible loads form a special class of parallelizable applications, which if given a large enough volume, can be *arbitrarily* partitioned into any number of independently- and identically-processable *load fractions*. Examples of applications that satisfy this divisibility property include massive dataset processing, image processing, signal processing, computation of Hough transforms, database search, simulations, and matrix computations. Divisible Load Theory (DLT) is the mathematical framework established to study Divisible Load Scheduling (DLS) [1–18, 20–27, 29, 30].

DLT has gained popularity because of its simplicity and deterministic nature. In a star connected network where the center of the star acts as the master and holds the entire load to be distributed, and the points of the star form the set of slave processors, the basic principle of DLT to determine an optimal schedule is the AFS (All nodes Finish Simultaneously) policy [3]. This states that in the optimum schedule, the load is distributed such that all the nodes involved in the computation finish processing their individual load fractions at the same time.

In the AFS policy, after the nodes finish computing their individual load fractions, no results are returned to the source. In most practical applications this is an unrealistic assumption, and the result collection phase contributes significantly to the total execution time, unless each node returns a small constant result such as a floating point or boolean value or a database record. This is not that uncommon, but the other cases need due consideration. All papers that have addressed result collection to date, have advocated simplistic LIFO (Last In, First Out) and FIFO (First In, First Out) sequences or variants thereof as solutions [1, 3, 7–9, 12, 17, 28]. It has been proved in [9] that LIFO and FIFO are not always optimal, and as this paper shows, these algorithms result in large errors in execution time as compared to the optimal schedule when the degree of node heterogenity is high.

Several papers have dealt with DLS on heterogeneous systems to date [2, 7–9, 12, 18, 28]. As far as can be judged, no paper has given a satisfactory solution to the scheduling problem where both the network bandwidth and computation capacities of the nodes are different, and the result transfer to the source is explicitly considered, i.e., to the DLS with Result Collection on HETerogeneous Systems (DLSRCHETS) problem. Along with the AFS policy, there are two assumptions that have implicitly pervaded DLT literature to date: (a) load is allocated to *all* processors, and (b) processors are never idle. The presence of idle time in the optimal schedule, which is a very important issue, has been overlooked in DLT work on result collection and heterogeneity. For the first time, [7, 8] proved that the optimal FIFO schedule can have a single processor with idle time, and that this processor can always be chosen to be the one to which load is allocated last.

In this paper, the completely general form of DLSRCHETS is tackled, with no assumptions being made regarding the number of processors allocated load, the network and node heterogeneity, or on the presence (or absence) of idle time. The main contribution of this paper is the new ITERLP algorithm and its rigorous testing through simulation. ITERLP does not necessarily use all processors

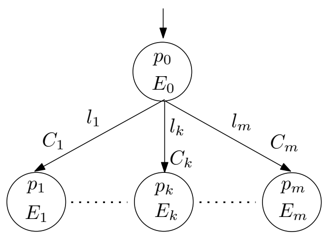

Figure 1. Heterogeneous star network $\mathcal{H}$

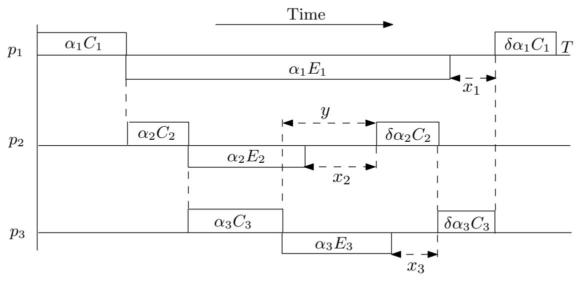

Figure 2. A feasible schedule for $m = 3$

and determines the number of processors to be used based on the system parameters (computation and communication capacities). The complexity of ITERLP is polynomial in the number of processors (m) though it requires solving $O(m^3)$ linear programs in the worst case.

The rest of the paper is as follows. In Sect. 2, the system model and the DLSRCHETS problem is described. Section 3 gives the ITERLP algorithm and simulation results are presented in Sect. 4. Section 5 provides the conclusion and future work.

2. System Model and Problem Definition

The divisible load $\mathcal{J}$ is to be distributed and processed on a heterogeneous star network $\mathcal{H} = (\mathcal{P}, \mathcal{L}, \mathcal{E}, \mathcal{C})$ as shown in Fig. 1, where $\mathcal{P} = \{p_0, \ldots, p_m\}$ is the set of $m+1$ processors, and $\mathcal{L} = \{l_1, \ldots, l_m\}$ is the set of m network links that connect the master scheduler (source) p_0 at the center of the star, to the slave processors $p_1, \ldots, p_m$. $\mathcal{E} = \{E_1, \ldots, E_m\}$ is the set of computation parameters of the slave processors, and $\mathcal{C} = \{C_1, \ldots, C_m\}$ is the set of communication parameters of the network links. E_k is the reciprocal of the speed of processor p_k, and C_k is the reciprocal of the bandwidth of link l_k. Both are defined in time units per unit load, i.e., p_k takes E_k time units to process a unit load transmitted to it from p_0 in C_k time units over the link l_k. It follows that $\forall k \in \{1, \ldots, m\} : \ E_k > 0, C_k > 0$.

The values in $\mathcal{E}$ and $\mathcal{C}$ are deterministic and available at the source. Based on these parameter values, the source p_0 splits $\mathcal{J}$ into parts (fractions) $\alpha_1, \ldots, \alpha_m$ and sends them to the respective processors $p_1, \ldots, p_m$ for computation. Each such set of m fractions is known as a *load distribution* $\alpha = \{\alpha_1, \ldots, \alpha_m\}$. The source does not retain any part of the load for computation. If it does, then it can be modeled as an additional slave processor with computation parameter E_0 and communication parameter $C_0 = 0$.

All processors follow a *single-port and no-overlap* model, implying that processors can communicate with only one other processor at a time, and communication and computation *cannot* occur simultaneously. The processors are continuously and exclusively available during the entire operation. The time taken for computation and communication is a linearly increasing function of the size of data.

The execution of the divisible load on each processor comprises of three distinct phases - the allocation phase, the computation phase, where the data is processed, and the result collection phase. The computation phase begins only after the entire load fraction allocated to that processor is received from the source. Similarly, the result collection phase begins only after the entire load fraction has been processed, and is ready for transmission back to the source (see Fig. 2). The source receives results from the child processors only after the entire load is distributed first.

For the divisible loads under consideration, such as image and video processing, Kalman filtering, matrix conversions, etc., the computation phase usually involves simple linear transformations, and the volume of returned results can be considered to be proportional to the amount of load received in the allocation phase. This is the accepted model for returned results in literature to date, [1,3,7–9,12,18,28,32]. If the allocated load fraction is α_k, then the returned result is equal to $\delta\alpha_k$, where $0 \leq \delta \leq 1$. The constant δ is application specific, and is the same for all processors for a particular load $\mathcal{J}$. For a load fraction α_k, $\alpha_k C_k$ is the transmission time from p_0 to p_k, $\alpha_k E_k$ is the computation time on p_k, and $\delta\alpha_k C_k$ is the time it takes p_k to finally transmit the results back to p_0.

Though a linear model is considered for computation and communication times for the sake of simplicity, all results can be easily extended to other (e.g. affine) cost models. For example, the computation time and the load allocation time of a processor p_k can be defined as $E_k(\alpha_k)$ and $C_k(\alpha_k)$, where $E_k(\cdot)$ and $C_k(\cdot)$ are functions of the allocated load fraction α_k. Similarly, the size of result data can be an application-dependent function $D(\cdot)$ of α_k, giving a result collection time $C_k(D(\alpha_k))$ for a processor p_k.

Usually the functions $C_k(\cdot)$, $E_k(\cdot)$, and $D(\cdot)$ will be concave, monotonically nondecreasing functions of α_k such as linear, affine, and constant functions.

σ_a and σ_c are two permutations of order m that represent the allocation and collection sequences respectively, i.e., $\sigma_a[k]$ and $\sigma_c[k]$ denote the processor number that occurs at index $k \in \{1, \ldots, m\}$. $\sigma_a(l)$ and $\sigma_c(l)$ are two *lookup func-*

tions that return the index of the processor $l \in \{1, \ldots, m\}$ in the allocation and collection sequences respectively. DLSRCHETS is defined as a linear program as below:

DLSRCHETS (DLS WITH RESULT COLLECTION ON HETEROGENEOUS SYSTEMS)

Given a heterogeneous network $\mathcal{H} = (\mathcal{P}, \mathcal{L}, \mathcal{E}, \mathcal{C})$, and a divisible load $\mathcal{J}$, find the sequence pair (σ_a, σ_c), and load distribution $\alpha = \{\alpha_1, \ldots, \alpha_m\}$ that

MINIMIZE $\zeta = 0\alpha_1 + \ldots + 0\alpha_m + T$

SUBJECT TO:

$$\sum_{j=1}^{\sigma_a(k)} \alpha_{\sigma_a[j]} C_{\sigma_a[j]} + \alpha_k E_k + \sum_{j=\sigma_c(k)}^{m} \delta \alpha_{\sigma_c[j]} C_{\sigma_c[j]} \leq T \qquad k = 1, \ldots, m \quad (1)$$

$$\sum_{j=1}^{m} \alpha_{\sigma_a[j]} C_{\sigma_a[j]} + \sum_{j=1}^{m} \delta \alpha_{\sigma_c[j]} C_{\sigma_c[j]} \leq T \quad (2)$$

$$\sum_{j=1}^{m} \alpha_j = \mathcal{J} \quad (3)$$

$$T \geq 0, \quad \alpha_k \geq 0 \qquad k = 1, \ldots, m \quad (4)$$

In the above formulation, for a triple $(\sigma_a, \sigma_c, \alpha)$, the LHS (Left Hand Side) of constraint (1) indicates the total time spent in transmission of tasks to all the processors that must receive load before the processor p_i can begin processing its allocated task, the computation time on the processor p_i itself, and the time for transmission back to the source of results of processor p_i, and all its subsequent result transfers. For the no-overlap model to be satisfied, the processing time T should be greater than or equal to this time for all the m processors. The single-port communication model is enforced by (2) since its LHS represents the lower bound on the time for distribution and collection under this model. The fact that the entire load is distributed amongst the processors is ensured by (3). This is known as the *normalization equation*. The non-negativity of the decision variables is ensured by constraint (4).

A linear programming problem in this form for a pair (σ_a, σ_c) can be solved using standard linear programming techniques in polynomial time [31]. There are $m!$ possible permutations each of σ_a and σ_c, and the linear program has to be evaluated $(m!)^2$ times to determine the globally optimal solution $(\sigma_a^*, \sigma_c^*, \alpha^*)$. Clearly, this is impractical to perform for more than a few processors. For example, it takes about 80 minutes for a PowerMac G5, with 2 GB of memory, to compute the optimal solution for 6 processors.

3. The ITERLP Algorithm

The ITERLP heuristic algorithm finds a solution by iteratively solving linear programs as follows. Processors are first sorted by increasing value of C_k (i.e., decreasing value of communication link bandwidth). The first two processors are selected and the optimal (σ_a, σ_c) pair (the one with the lowest processing time for the two processors) is determined by solving the linear program defined by the constraints (1) to (4) four times for each permutation of the allocation and collection sequence. The next processor in the sequence is added in the next iteration and the linear program is solved again. The new processor can be interleaved at any position in (σ_a, σ_c), but with an additional constraint that the relative positions of processors already determined are maintained. By constraining the number of possible sequences in this manner, if the number of processors in an iteration is k, $k \leq m$, then k^2 linear programs are solved in that iteration instead of the possible $(k!)^2$.

For example, if the optimal sequences at the end of the first iteration are $\sigma_a^1 = \{1, 2\}$ and $\sigma_c^1 = \{2, 1\}$, then in the second iteration, the set of possible allocation sequences is $\Sigma_a^2 = \{(3,1,2), (1,3,2), (1,2,3)\}$, and the set of possible collection sequences is $\Sigma_c^2 = \{(3,2,1), (2,3,1), (2,1,3)\}$.

In any iteration, if processor k is allocated zero load, then the algorithm terminates and does not proceed to the next iteration with $k+1$ processors. In the worst case, $\sum_{k=1}^{m} k^2 = O(m^3)$ linear programs have to be solved in the ITERLP algorithm. To compare performance with OPT for example, on the same machine as described in Sect. 2, ITERLP can find the solution for about 65 processors in 80 minutes, but when m is increased to 100, it takes around 15 hours. Of course, this is much too expensive to be practically used for large values of m. However, it is found that ITERLP generates significantly better schedules than traditional algorithms (see Sect. 4) and it can be used as a benchmark to compare other heuristic algorithms.

The rationale behind ITERLP is as follows. All optimality results to date for DLS on heterogeneous systems, those ignoring result collection [10, 12, 13, 16, 22, 23] as well as those considering result collection [1, 3, 7–9], have advocated load allocation in the order of decreasing communi-

Table 1. Results for $\mathcal{C} = \{10, 15\}$, $\mathcal{E} = \{10, 10\}$, $\delta = 0.5$

Algorithm	σ_a	σ_c	α	T
OPT	$\{1,2\}$	$\{1,2\}$	$\{0.625, 0.375\}$	18.4375
ITERLP	$\{1,2\}$	$\{1,2\}$	$\{0.625, 0.375\}$	18.4375
LIFOC	$\{1,2\}$	$\{2,1\}$	$\{0.765, 0.235\}$	19.1176
FIFOC	$\{1,2\}$	$\{1,2\}$	$\{0.625, 0.375\}$	18.4375

Table 2. Results for $\mathcal{C} = \{10, 15, 20\}$, $\mathcal{E} = \{10, 10, 1\}$, $\delta = 0.5$

Algorithm	σ_a	σ_c	α	T
OPT	$\{1,2,3\}$	$\{3,2,1\}$	$\{0.7108, 0.2187, 0.0705\}$	17.7690
ITERLP	$\{1,2,3\}$	$\{3,1,2\}$	$\{0.6126, 0.3676, 0.0198\}$	18.0371
LIFOC	$\{1,2,3\}$	$\{3,2,1\}$	$\{0.7108, 0.2187, 0.0705\}$	17.7690
FIFOC	$\{1,2,3\}$	$\{1,2,3\}$	$\{0.6061, 0.3636, 0.0303\}$	18.1818

cation link bandwidth. Hence processors are initially sorted in that order in ITERLP. Since neither LIFO nor FIFO schedules are always optimal, the new processor being introduced in an iteration could potentially be interleaved in any position in the optimal sequence. So the ITERLP heuristic tests all possible positions for the newly introduced processor. To build the sequences at reasonable (polynomial) cost, ITERLP assumes that the relative positions of the processors already determined are not modified by the additional processor. The following example with three processors proves that this is not true in general. Nevertheless, as the simulation results in the next section prove, ITERLP produces near-optimal results in general, at a reasonable cost.

Let $\mathcal{C} = \{10, 15, 20\}$, $\mathcal{E} = \{10, 10, 1\}$, and $\delta = 0.5$. The results obtained for the different algorithms are given in Tables 1 and 2. Details of the algorithms used are given in Sect 4. It is observed that after the first iteration, the optimal sequences found by ITERLP are $\sigma_a^1 = \{1, 2\}$ and $\sigma_c^1 = \{1, 2\}$. In the second iteration, when processor p_3 is added, ITERLP returns the sequences as $\sigma_a^2 = \{1, 2, 3\}$ and $\sigma_c^2 = \{3, 1, 2\}$. However, the optimal sequences for the three processors are $\sigma_a^* = \{1, 2, 3\}$ and $\sigma_c^* = \{3, 2, 1\}$. That is, the optimal collection sequence for the first two processors is reversed by the addition of the third processor. However, to date no set of values of $\mathcal{C}$, $\mathcal{E}$, and δ have been found that reverse the order of processors' allocation sequence in the optimal schedule. The allocation sequence in the order of decreasing communication bandwidth is always found optimal.

4. Simulation Results and Analysis

In the simulations, all E_k and C_k are defined in the same time units. On open networks, it is not unusual for processors to have widely varying values of E_k and C_k, with the ratios $\min(E_k) : \max(E_k)$ and $\min(C_k) : \max(C_k)$ reaching 1:100 [19]. Further, they can appear in any combination. For example, a fast processor may have a very slow network connection, while a processor with a fast link may be overloaded and not have enough computation speed. Along with system heterogeneity, it is important to verify the effect of the application on the algorithms. To rigorously test the performance of ITERLP, several simulations were performed with different ranges for E_k, C_k, and δ.

The performance of ITERLP was compared to three algorithms, viz. OPT, FIFOC, and LIFOC, which are explained below. In all, more than 300,000 simulation runs were carried out using parameter values that cover most situations observed in practice.

The globally optimal schedule OPT is obtained after evaluation of the linear program for all possible $(m!)^2$ permutations of (σ_a, σ_c). The MATLAB$^{\text{TM}}$ linear programming solver `linprog` is used to determine the optimal solution (load distribution) for each permutation pair. The processing time for each pair is calculated, and the sequence pair and load distribution that results in the minimum processing time is selected as the OPT solution.

FIFOC and LIFOC distribute load fractions in the order of decreasing communication link bandwidth (i.e., increasing value of communication parameter, C_k). The sequence of result collection for FIFOC is the same as the sequence of allocation, while the result collection sequence of LIFOC is in the reverse order of the sequence of allocation. For the pair (σ_a, σ_c) so obtained, `linprog` is used to determine the optimal solution (load distribution) and processing time.

Preliminary simulations for other heuristic algorithms, viz. FIFO, LIFO, FIFOE, LIFOE, and SUMCE, revealed such large errors in favor of ITERLP, that it was decided not to pursue them further. The solutions to FIFO and LIFO are calculated similar to FIFOC and LIFOC except for the fact that the processors are not initially sorted. FIFOE and LIFOE

Table 3. Parameters for Simulation

Case	$C_k \in$	$E_k \in$	Case	$C_k \in$	$E_k \in$
1	[1,10]	[1,10]	14	[10,100]	[1,100]
2	[1,10]	[10,100]	15	[10,100]	[10,1000]
3	[1,10]	[100,1000]	16	[10,1000]	[1,10]
4	[1,10]	[1,100]	17	[10,1000]	[10,100]
5	[1,10]	[10,1000]	18	[10,1000]	[100,1000]
6	[1,100]	[1,10]	19	[10,1000]	[1,100]
7	[1,100]	[10,100]	20	[10,1000]	[10,1000]
8	[1,100]	[100,1000]	21	[100,1000]	[1,10]
9	[1,100]	[1,100]	22	[100,1000]	[10,100]
10	[1,100]	[10,1000]	23	[100,1000]	[100,1000]
11	[10,100]	[1,10]	24	[100,1000]	[1,100]
12	[10,100]	[10,100]	25	[100,1000]	[10,1000]
13	[10,100]	[100,1000]			

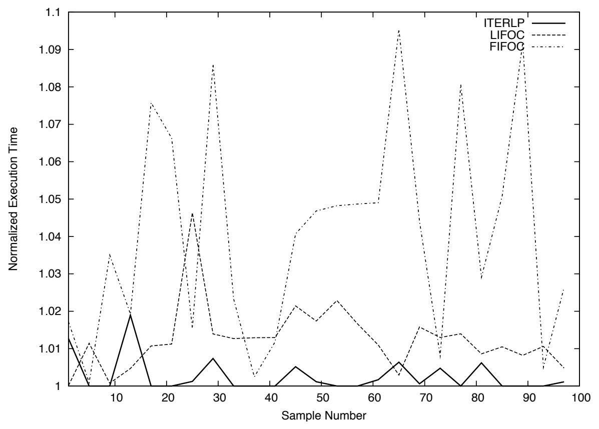

Figure 3. Normalized Execution Time for $m = 4$, $\delta = 0.2$, Case 9

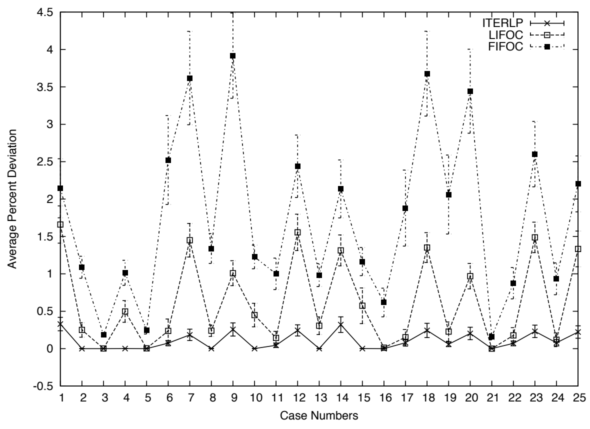

Figure 4. $\langle \Delta T \rangle$ for $m = 4$, $\delta = 0.2$

Table 4. max $\langle \Delta T \rangle$ of FIFOC

m	$\delta = 0.2$			$\delta = 0.5$			$\delta = 0.8$		
	FIFOC	#	ITERLP	FIFOC	#	ITERLP	FIFOC	#	ITERLP
4	3.91	9	0.25	10.29	7	0.24	12.31	18	0.35
5	4.40	7	0.32	10.96	7	0.35	14.33	7	0.36

Table 5. max $\langle \Delta T \rangle$ of LIFOC

m	$\delta = 0.2$			$\delta = 0.5$			$\delta = 0.8$		
	LIFOC	#	ITERLP	LIFOC	#	ITERLP	LIFOC	#	ITERLP
4	1.66	1	0.33	1.39	23	0.45	1.13	23	0.53
5	1.24	23	0.37	1.27	4	0.08	1.63	4	0.12

distribute load fractions in the order of decreasing computation speed (i.e., increasing value of computation parameter, E_k). SUMCE distributes and collects load fractions in the order of increasing value of the sum $C_k + E_k + \delta C_k$ (equivalent to sorting by the sum $C_k + E_k$).

Simulations were carried out for $m = 4, 5$ and $\delta = 0.2, 0.5, 0.8$. For each variant algorithm, viz. OPT, LIFOC, FIFOC, and ITERLP, at each value of m and δ, 100 simulation runs were carried out for the 25 cases in Table 3. The values of E_k and C_k were obtained by sampling continuous uniform distributions in the regions specified in Table 3. The total processing time for each variant algorithm, T_{VARIANT}, was calculated in each run. For example, Fig. 3 shows the execution times normalized with respect to T_{OPT} for $m = 4$, $\delta = 0.2$, case number 9. The solid line indicates the performance of ITERLP. As can be observed, ITERLP has the best performance, followed by LIFOC and FIFOC.

Fig. 3 also distinctly shows the dependence of processing time on the system parameters. To quantify the performance of the algorithms, the percentage deviation from the optimal processing time, $\Delta T_{\text{VARIANT}}$, for each variant in each of the 25 cases was calculated as:

$$\Delta T_{\text{VARIANT}} = \frac{T_{\text{VARIANT}} - T_{\text{OPT}}}{T_{\text{OPT}}} * 100\% \quad (5)$$

Mean deviation from optimal, $\langle \Delta T \rangle$, for each variant was calculated by averaging ΔT over 100 simulation runs. Values of $\langle \Delta T \rangle$ were then plotted.

The plot for $m = 4$, $\delta = 0.2$ is shown in Fig. 4. It can be observed that ITERLP consistently outperforms FIFOC and LIFOC in all the cases. As the value of δ increases, it is observed that the performance of LIFOC and ITERLP becomes very similar, while the error of FIFOC increases, as the plot for $m = 5$, $\delta = 0.8$ in Fig. 5 shows. Not only does the performance become similar, but also it gets very close to optimal. It can be concluded that for heterogeneous systems, where result collection time is large (comparable to the load allocation time), the performance of LIFOC and ITERLP is almost equal and optimal.

For intermediate values of δ, the performance of ITERLP is moderately better than LIFOC, and largely better than FIFOC as seen in Fig. 6 for $m = 5$, $\delta = 0.5$.

Though the algorithms show a clear dependence on the value of δ, the reason for the variation in performance can only be hypothesized at this juncture. In the case of LIFOC for example, when δ is large ($\delta >> 1$), and especially when $\delta \to +\infty$, the load allocation and result collection looks similar to the case when $\delta = 0$, only in the reverse. LIFOC is still optimal in this case (if it was optimal earlier), while for FIFOC it would be the worst possible sequence. For the case when $\delta = 1$, by using the schedule transformation explained in [9], it can be seen that LIFOC processes exactly twice the amount of load in half the time, as would be processed if there would not have been any result collection phase. No such statement can be made about FIFOC.

Table 4 gives the maximum values of $\langle \Delta T \rangle$ for FIFOC, the case numbers when they occur, and the corresponding

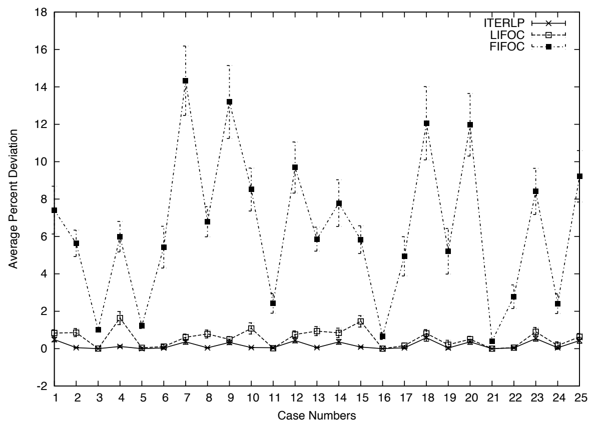

Figure 5. $\langle \Delta T \rangle$ **for** $m = 5$**,** $\delta = 0.8$

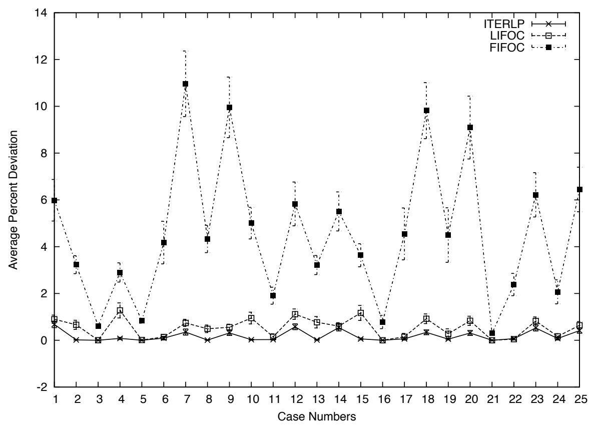

Figure 6. $\langle \Delta T \rangle$ **for** $m = 5$**,** $\delta = 0.5$

values of $\langle \Delta T \rangle$ for ITERLP for those cases. Table 5 gives similar values for LIFOC. The value of $\langle \Delta T \rangle$ for FIFOC for $m = 4$, $\delta = 0.5$, case number 7, is 43 times that of ITERLP for that case. Similarly, $\langle \Delta T \rangle$ for LIFOC for $m = 5$, $\delta = 0.8$, case number 4, is 13.5 times that of ITERLP for that case. That is, ITERLP generates good schedules for cases that cause FIFOC and LIFOC to perform poorly.

The maximum values of $\langle \Delta T \rangle$ of ITERLP and the case numbers when they occur are given in Table 6. The maximum $\langle \Delta T \rangle$ of ITERLP is 0.68%, and it occurs at case number 1 of $m = 5$, $\delta = 0.5$. It is observed that the error remains below 1% irrespective of the value of m or δ.

To evaluate the performance of the algorithms with the increase in number of nodes, the processing times of FIFOC and LIFOC were compared with ITERLP. This is because, OPT cannot be practically carried out beyond $m = 5$. 100 simulation runs were carried out for $m = 10, 20, 30$, $\delta = 0.2, 0.5, 0.8$ for each of the 25 cases listed in Table 3.

As the number of processors and the value of δ increase, the performance of ITERLP and LIFOC becomes very similar, while there is an increase in the error of FIFOC. The 95% confidence interval bars indicate that the $\langle \Delta T \rangle$ of FIFOC with respect to ITERLP varies widely. The progression of performance is clearly reflected in Figs. 7 to 9 that plot the values of $\langle \Delta T \rangle$ with respect to ITERLP for (m, δ) pairs $(10, 0.2)$, $(20, 0.5)$, and $(30, 0.8)$ respectively.

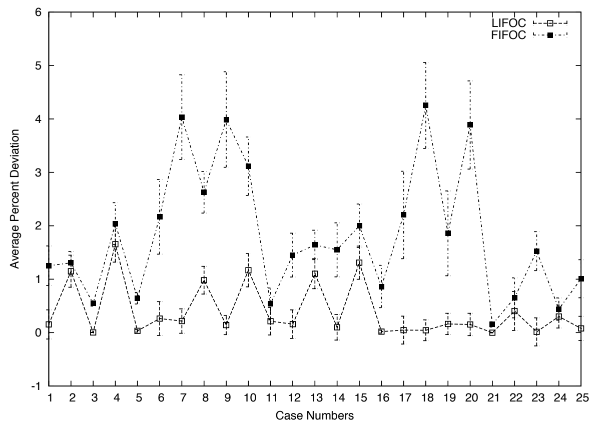

Figure 7. $\langle \Delta T \rangle$ **w.r.t.** ITERLP**,** $m = 10$**,** $\delta = 0.2$

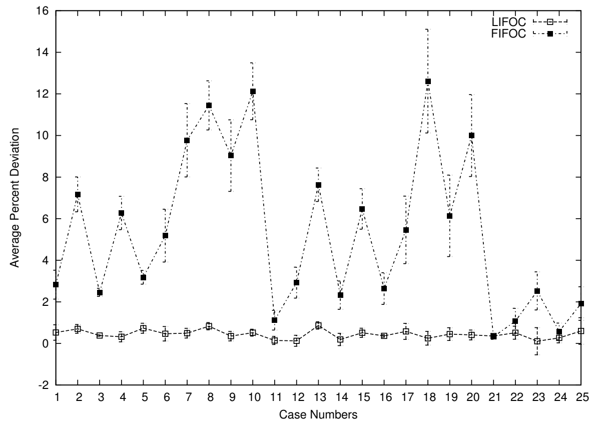

Figure 8. $\langle \Delta T \rangle$ **w.r.t.** ITERLP**,** $m = 20$**,** $\delta = 0.5$

However, at small values of δ, ITERLP performs better than both LIFOC and FIFOC even with large number of processors as can be seen in Fig. 10 that plots the value of $\langle \Delta T \rangle$ with respect to ITERLP for $m = 30$, $\delta = 0.2$.

5. Conclusion

In this paper, a new heuristic algorithm, ITERLP, for the scheduling of divisible loads on heterogeneous systems and considering the result collection phase is presented. A large number of simulations are performed and it is found that ITERLP consistently delivers near-optimal performance irrespective of the degree of heterogeneity of the system, the number of nodes, or the size of result data. Instead of having

Table 6. max $\langle \Delta T \rangle$ of ITERLP

m	$\delta = 0.2$				$\delta = 0.5$				$\delta = 0.8$			
	ITERLP	case	LIFOC	FIFOC	ITERLP	case	LIFOC	FIFOC	ITERLP	case	LIFOC	FIFOC
4	0.33	1	1.66	2.14	0.45	23	1.39	5.81	0.53	23	1.13	7.35
5	0.51	12	1.13	2.41	0.68	1	0.90	5.99	0.58	18	0.82	12.06

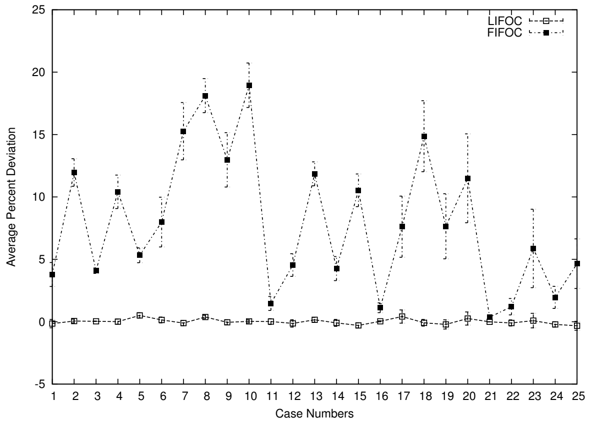

Figure 9. $\langle \Delta T \rangle$ **w.r.t.** ITERLP, $m = 30$, $\delta = 0.8$

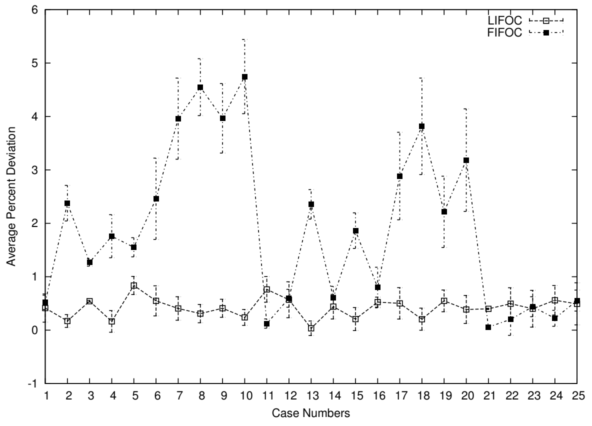

Figure 10. $\langle \Delta T \rangle$ **w.r.t.** ITERLP, $m = 30$, $\delta = 0.2$

a single predefined sequence, ITERLP iteratively builds a locally optimal solution leading to a low error value. ITERLP is much too expensive to be practically used for large values of m. But as it generates significantly better schedules than traditional algorithms in general, it can be used as a benchmark to compare other heuristic algorithms.

As future work, an algorithm with similar performance, but with better cost characteristics than ITERLP needs to be found. Another important area would be to extend the results to multi-level processor trees.

References

[1] M. Adler, Y. Gong, and A. L. Rosenberg. Optimal sharing of bags of tasks in heterogeneous clusters. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10, New York, NY, USA, 2003. ACM.

[2] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Trans. Parallel Distrib. Syst.*, 15(4):1–12, April 2004.

[3] G. D. Barlas. Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees. *IEEE Trans. Parallel Distrib. Syst.*, 9(5):429–441, May 1998.

[4] S. Bataineh and B. Al-Asir. An efficient scheduling algorithm for divisible and indivisible tasks in loosely coupled multiprocessor systems. *Software Engineering Journal*, 9(1):13–18, Jan. 1994.

[5] S. Bataineh, T.-Y. Hsiung, and T. G. Robertazzi. Closed form solutions for bus and tree networks of processors load sharing a divisible job. *IEEE Trans. Comput.*, 43(10):1184–1196, Oct. 1994.

[6] S. Bataineh and T. G. Robertazzi. Performance limits for processors with divisible jobs. *IEEE Trans. Aerosp. Electron. Syst.*, 33(4):1189–1198, Oct. 1997.

[7] O. Beaumont, L. Marchal, V. Rehn, and Y. Robert. FIFO scheduling of divisible loads with return messages under the one-port model. *Research Report 2005-52*, Oct. 2005.

[8] O. Beaumont, L. Marchal, V. Rehn, and Y. Robert. FIFO scheduling of divisible loads with return messages under the one port model. In *Proc. Heterogeneous Computing Workshop HCW'06*, Apr. 2006.

[9] O. Beaumont, L. Marchal, and Y. Robert. Scheduling divisible loads with return messages on heterogeneous master-worker platforms. *Research Report 2005-21*, May 2005.

[10] V. Bharadwaj, D. Ghose, and V. Mani. Optimal sequencing and arrangement in distributed single-level tree networks with communication delays. *IEEE Trans. Parallel Distrib. Syst.*, 5(9):968–976, Sept. 1994.

[11] V. Bharadwaj, D. Ghose, and V. Mani. Multi-installment load distribution in tree networks with delays. *IEEE Trans. Aerosp. Electron. Syst.*, 31(2):555–567, Apr. 1995.

[12] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[13] V. Bharadwaj, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, Jan. 2003.

[14] V. Bharadwaj, X. Li, and C. C. Ko. Efficient partitioning and scheduling of computer vision and image processing data on bus networks using divisible load analysis. *Image and Vision Computing*, 18(1):919–938, Jan. 2000.

[15] V. Bharadwaj, X. Li, and C. C. Ko. On the influence of start-up costs in scheduling divisible loads on bus networks. *IEEE Trans. Parallel Distrib. Syst.*, 11(12):1288–1305, Dec. 2000.

[16] J. Blazewicz and M. Drozdowski. Distributed processing of divisible jobs with communication startup costs. *Discrete Applied Mathematics*, 76(1-3):21–41, June 1997.

[17] Y.-C. Cheng and T. G. Robertazzi. Distributed computation for a tree network with communication delays. *IEEE Trans. Aerosp. Electron. Syst.*, 26(3):511–516, May 1990.

[18] N. Comino and V. L. Narasimhan. A novel data distribution technique for host-client type parallel applications. *IEEE Trans. Parallel Distrib. Syst.*, 13(2):97–110, Feb. 2002.

[19] A. Galstyan, K. Czajkowski, and K. Lerman. Resource Allocation in the Grid Using Reinforcement Learning. In *Intl. Jt. Conf. on Autonomous Agents and Multiagent Systems*, volume 3, pages 1314–1315, 2004.

[20] D. Ghose and H. J. Kim. Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays. *Journal of Parallel and Distributed Computing*, 55(1):32–59, Nov. 1998.

[21] D. Ghose and V. Mani. Distributed computation with communication delays: Asymptotic performance analysis. *Journal of Parallel and Distributed Computing*, 23(3):293–305, Dec. 1994.

[22] H. J. Kim, G. in Jee, and J. G. Lee. Optimal load distribution for tree network processors. *IEEE Trans. Aerosp. Electron. Syst.*, 32(2):607–612, Apr. 1996.

[23] B. Kreaseck, L. Carter, H. Casanova, and J. Ferrante. Autonomous protocols for bandwidth-centric scheduling of independent-task applications. *Proc. International Parallel and Distributed Processing Symposium, 2003.*, pages 10 pp.–, Apr. 2003.

[24] C.-H. Lee and K. G. Shin. Optimal task assignment in homogeneous networks. *IEEE Trans. Parallel Distrib. Syst.*, 8(2):119–129, Feb. 1997.

[25] X. Li, V. Bharadwaj, and C. C. Ko. Divisible load scheduling on single-level tree networks with buffer constraints. *IEEE Trans. Aerosp. Electron. Syst.*, 36(4):1298– 1308, Oct. 2000.

[26] V. Mani and D. Ghose. Distributed computation in linear networks: Closed-form solutions. *IEEE Trans. Aerosp. Electron. Syst.*, 30(2):471–483, Apr. 1994.

[27] T. G. Robertazzi. Processor equivalence for daisy chain load sharing processors. *IEEE Trans. Aerosp. Electron. Syst.*, 29(4):1216–1221, Oct. 1993.

[28] A. Rosenberg. Sharing partitionable workload in heterogeneous NOWs: Greedier is not better. In *IEEE International Conference on Cluster Computing*, pages 124–131, Newport Beach, CA, October 2001.

[29] J. Sohn and T. G. Robertazzi. Optimal divisible job load sharing for bus networks. *IEEE Trans. Aerosp. Electron. Syst.*, 32(1):34–40, Jan. 1996.

[30] J. Sohn, T. G. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Trans. Parallel Distrib. Syst.*, 9(3):225–234, Mar. 1998.

[31] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*, volume 37 of *International Series in Operations Research & Management*. Kluwer Academic Publishers, 2nd edition, 2001.

[32] D. Yu and T. G. Robertazzi. Divisible load scheduling for grid computing. In *Proc. International Conference on Parallel and Distributed Computing Systems (PDCS 2003)*, volume 1, Los Angeles, CA, USA, Nov. 2003.

Biographies

Abhay Ghatpande received his B. Engineering degree from University of Pune, India in 1997, and his M.S. degree from Waseda University, Tokyo in 2004. He is presently a Research Associate and Ph.D. candidate there. His research interests include parallel and distributed computing, multi-agent systems, and use of inference and learning algorithms in high performance computing. He is a member of the IEEE and IEICE.

Olivier Beaumont received his Ph.D from the University of Rennes in 1999. From 1999 to 2001, he was assistant professor at Ecole Normale Suprieure de Lyon. He was appointed at ENSEIRB in Bordeaux. In 2004, he defended his "habilitation diriger les recherches". He is the author of more than 15 papers published in international journals and 50 papers published in international conferences. His research interests are design of parallel, distributed and randomized algorithms, overlay networks on large scale heterogeneous platforms and combinatorial optimization.

Hidenori Nakazato received his B. Engineering degree in Electronics and Telecommunications from Waseda University in 1982 and his M.S. and Ph.D. in computer science from Univ. of Illinois in 1989 and 1993, respectively. He was with Oki Electric from 1982 to 2000. Since 2000, he has been a professor at Graduate School of Global Information and Telecommunications Studies, Waseda University. His research interests include performance issues in distributed systems.

Hiroshi Watanabe received the B.E., M.E. and Ph.D. degrees from Hokkaido University, Japan, in 1980, 1982 and 1985, respectively. He joined NTT Corporation in 1985, and engaged in R&D for image and video coding at NTT Human Interface Labs until Aug 2000. He also engaged in developing JPEG, MPEG standards under JTC 1/SC 29. In Sept 2000, he moved to Waseda University, as a professor at the Graduate School of Global Information and Telecommunication Studies. He was ISO/IEC JTC 1/SC 29 Chairman from 1999 to 2006. He was an associate editor of IEEE Trans. of Circuits and Systems for Video Technology. He is a member of IEEE, IEICE, IPSJ, ITE, IIEEJ.