IEICE TRANS. COMMUN., VOL.E91–B, NO.7 JULY 2008

PAPER

Analysis of Divisible Load Scheduling with Result Collection on Heterogeneous Systems

Abhay GHATPANDE[†a)], ***Student Member***, **Hidenori NAKAZATO**[†], ***Member***,
Olivier BEAUMONT[††], ***Nonmember***, ***and*** **Hiroshi WATANABE**[†], ***Member***

SUMMARY Divisible Load Theory (DLT) is an established framework to study Divisible Load Scheduling (DLS). Traditional DLT ignores the *result collection phase*, and specifies no solution to the general case where both the network speed and computing capacity of the nodes are heterogeneous. In this paper, the DLS with Result Collection on HETerogeneous Systems (DLSRCHETS) problem is formulated as a linear program and analyzed. The papers to date that have dealt with result collection, proposed simplistic LIFO (Last In, First Out) and FIFO (First In, First Out) type of schedules as solutions. The main contributions of this paper are: (a) A proof of the *Allocation Precedence Condition*, which is inconsequential in LIFO or FIFO, but is important in a general schedule. (b) A proof of the *Idle Time Theorem*, which states that irrespective of whether load is allocated to all available processors, in the optimal solution to the DLSRCHETS problem, at the most one processor that is allocated load has idle time, and that the idle time exists only when the result collection begins immediately after the completion of load distribution.

key words: *divisible load scheduling, heterogeneous systems, result collection*

1. Introduction

Divisible loads are 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 data-set processing, image processing, signal processing, computation of Hough transforms, database search, simulations, and matrix computations.

With the proliferation of the Internet, *volunteer computing* or *desktop grid computing* is rapidly becoming feasible and gaining popularity. Volunteer computing is a form of distributed computing in which a large number of average users volunteer their computers to serve as processing and storage resources for scientific research projects [1]–[3]. Divisible loads are especially suited for volunteer computing because of the absence of interdependencies and precedence relations. Unlike other types of distributed computing, volunteer computing uses anonymous contributed resources, and consequently, a large degree of heterogeneity exists in the network bandwidth and processing power of the participating nodes.

Divisible Load Theory (DLT) is the mathematical framework that has been established to study Divisible Load Scheduling (DLS) [4]–[25]. The hallmark of DLT has been its relative simplicity and deterministic nature. In a star connected (single-level tree) network where the center of the star (root of the tree) forms the source and holds the entire load to be distributed, and the points of the star (leaf nodes of the tree) form the computing elements that process the load fractions allocated to them, the basic principle of DLT to determine an optimal schedule is the AFS (All nodes Finish Simultaneously) policy [20]. This states that the optimum schedule is one in which the load is distributed such that all the nodes involved in the computation finish processing their individual load fractions at the same time. This policy yields closed-form equations for the load fractions, and allows easy theoretical analysis.

The AFS policy implies that after the nodes finish computing their individual load fractions, no results are returned to the source. This is an unrealistic assumption for the applications in which the result collection phase contributes significantly to the total execution time. All papers that addressed result collection to date, have advocated simplistic FIFO (First In, First Out) and LIFO (Last In, First Out) schedules. In FIFO, results are collected in the same order as that of load allocation, while in LIFO, the order is reversed. It has been proved in [26] that LIFO and FIFO are not always optimal, and in our opinion, there is no compelling reason to use them over other possible sequences.

Some papers have dealt with heterogeneous systems. To the best of our knowledge, no paper has given a satisfactory solution to the case where both the network bandwidth and computation capacities of the nodes are different, and result transfer to the source is explicitly considered. Only a few of the papers that tackled heterogeneity have considered the possibility of idle time in the optimal schedule, or the fact that some processors may not be allocated load for processing.

In this paper, the DLS with Result Collection on HETerogeneous Systems (DLSRCHETS) problem is formulated and analyzed in detail. A completely general form of DLSRCHETS is tackled, with no assumptions being made regarding the number of processors allocated load, the network and computation heterogeneity, or on the presence (or absence) of idle time. The major contributions of this paper are: (a) A proof of the *Allocation Precedence Condition*, which ensures that there exists an optimal schedule in which the source distributes load to all the processors first, before re-

Manuscript received January 10, 2008.

[†]The authors are with Waseda University, Tokyo, 169-0051 Japan.

[††]The author is with INRIA Futurs — LaBRI, France.

a) E-mail: abhay@toki.waseda.jp

DOI: 10.1093/ietcom/e91–b.7.2234

Copyright © 2008 The Institute of Electronics, Information and Communication Engineers

ceiving any results. This condition is immaterial in LIFO and FIFO schedules, but is important in the general schedule. (b) A proof of the *Idle Time Theorem*, which states that irrespective of whether load is allocated to all available processors, in the optimal solution to the DLSRCHETS problem, at the most one of the processors that is allocated load has idle time, and that the idle time exists only when the result collection begins immediately after the completion of load distribution. Though linear models for computation and communication time are used in this paper for simplicity, all the results can be easily extended to affine cost models.

The rest of the paper is as follows. In Sect. 2 the results obtained to date related to DLSRCHETS are discussed. Section 3 provides a detailed description of the DLSRCHETS problem along with the proof of the allocation precedence condition. Section 4 analyzes the optimal solution of the DLSRCHETS problem, and proves the idle time theorem. Finally, Sect. 5 gives the conclusion and future work.

2. Related Work

To the best of our knowledge, no paper to date has comprehensively dealt with the issues of result collection and node heterogeneity considered together, nor has any polynomial time algorithm been proposed to find the optimum load allocation and result collection sequence in a heterogeneous network.

Bharadwaj, et al. [4, Chap. 5] proved that the sequence of allocation of data to the processors is important in heterogeneous networks. Without considering result collection, they proved that for optimum performance, (a) when processors have equal computation capacity, the optimal schedule results when the fractions are allocated in the order of decreasing communication link capacity, and (b) when communication capacity is equal, the data should be allocated in the order of decreasing computation capacity.

Cheng and Robertazzi [19] and Bharadwaj, et al. [4, Chap. 3] addressed the issue of result collection with a simplistic constant result collection time, which is possible only for a limited number of applications on homogeneous networks. Barlas [20] explicitly addressed the result collection phase for single-level and arbitrary tree networks, but an assumption regarding the absence of idle time was made without justification. Essentially only two cases were analyzed: (a) when communication overhead is zero, and (b) when communication networks are homogeneous. The optimal sequences derived were essentially LIFO or FIFO. Rosenberg [27] too proposed the LIFO and FIFO sequences for result collection. He concluded through simulations that FIFO is better when the communication network is homogeneous with a large number of processors, while LIFO is advantageous when the network is heterogeneous with a small number of processors.

Traditionally, DLT has focussed only on single-installment [4, Chap. 8] delivery of data. Banino, et al. [28] and Beaumont, et al. [29] considered a multi-installment strategy. The data is considered to be split into equal sized tasks, and the maximum number of tasks that can be delivered to the processors in a given time interval is found. They argue that in the *steady state*, separate modeling of result collection is unnecessary. They concluded that allocation should proceed in the order of decreasing communication bandwidth for optimal performance in the steady-state.

In this paper, the focus is on the more traditional form of single-installment DLS on account of the following reasons:

- To get a better understanding of the underlying problem structure when result collection and node heterogeneity are considered together.
- The scheduling of tasks is essentially left to chance in the multi-installment strategy. Collisions during communication are likely to cause delays.
- For certain applications, multi-installment distribution of data is difficult or not desirable, especially in cases where there is data interdependency.

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 after receiving their load fractions. 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, it has been shown in [26] that the LIFO and FIFO orderings are not always optimal for a given set of processors. In [30], [31], it has been stated that all processors from a given set of processors may not be used in the optimal solution. For the single-port communication model (see Sect. 3.1), [26], [30] and [31] proved the following features in optimal schedules:

- Assuming LIFO and FIFO orderings, load is allocated in the order of decreasing communication link bandwidth.
- LIFO ordering never has idle time in any processor.
- Assuming FIFO ordering, at the most one processor may have idle time.
- The processor with idle time in an optimal FIFO schedule can always be chosen to be the last processor in the allocation sequence (i.e. the processor with the slowest communication link).

We believe that the work presented in this paper is a *logical progression* to the results in [26], [30], [31]. The above mentioned optimality results have been derived strictly for LIFO and FIFO type of schedules. Since any one of the other possible processor orderings could be the optimal solution to the DLSRCHETS problem, the next logical question is, "Can these optimality results also be extended to the general case?" For example,

- Should all processors be allocated load first before they start sending results back to the source in the general case?
- [26], [30], [31] define the DLSRCHETS problem as a linear program only for a single pair of allocation and collection sequences. Can the general problem be defined in

a similar way? If so, what are the necessary conditions to be able to do that? (The above mentioned *allocation precedence condition* is one of them.)
- Will all available processors be allocated load in the optimal solution for the general case?
- How many processors out of those that are allocated load will have idle time in the optimal solution for the general case?

These are some of the questions that we attempt to answer in this paper.

3. Problem Description

In this paper, a divisible load refers to an *arbitrarily divisible* load [4, Chap. 1] that can be divided into any number of fractions of arbitrary size, without restriction on the granularity of division. Each fraction undergoes identical processing irrespective of its size, and there exist no precedence relations between the fractions, so that each fraction can be processed independently of the others.

3.1 System Model

The divisible load $\mathcal{J}$ is to be distributed and processed on a heterogeneous star network $\mathcal{H} = (\mathcal{P}, \mathcal{L}, \mathcal{E}, C)$ 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 $C = \{C_1, \ldots, C_m\}$ the set of communication parameters of the network links. The computation and communication parameters are the inverse of the speed of the processors and links respectively, and 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 is assumed that all processors are continuously and exclusively available, and have sufficient buffer capacity to receive the entire load fraction in a single installment from the source. The values in $\mathcal{E}$ and C are assumed to be deterministic and available at the source. Based on these 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 extra slave processor with computation parameter E_0 and communication parameter $C_0 = 0$.

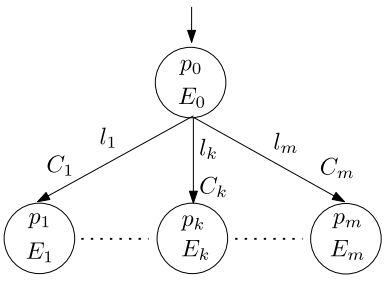

Fig. 1 Heterogeneous star network $\mathcal{H}$.

All processors follow a *single-port and no-overlap* communication model, implying that processors can communicate with only one other processor at a time, and communication and computation *cannot* occur simultaneously. A few papers have dealt with DLS with a *multi-port* model [32]–[34]. This model was first proposed for *Hypercubes* in [35]. If it is possible to have as many ports as there are slaves, and also be able to program the source to communicate simultaneously with all the slaves, then the problem of sequencing allocation and result collection becomes irrelevant. We use the *single-port* model for the following reasons:

- Traditionally, DLT has used the single-port (sequential) communication model, as evidenced by the large body of literature using this model mentioned in Sect. 1 and 2 versus the three papers [32]–[34] cited above for the multi-port model.
- As mentioned in Sect. 1, this paper addresses DLS on generic heterogeneous systems such as the Internet and volunteer computing platforms. The master-slave topology is an application-level *logical construct* on these systems. The source is not a special machine as used in the papers [32]–[35] referenced above, but can be any machine that wants to participate in the computation.
- An experimental setup such as the one described in [30], [31] using MPI (Message-Passing Interface) to implement the master-slave processing follows the single-port model as it is found to be more realistic in practice. As noted in [36], *scatter-gather* operations in MPI need to be improved before it can be reliably used for simultaneous data transfer to and from several slaves.
- If each slave was to be connected to the master by a dedicated link (port), then the number of slave processors that could be used would be seriously limited as it is not practical to have a large number of physical ports on a computer.

The execution of the divisible job on each processor comprises of three distinct phases — the allocation phase, where data is sent to the processor from the source, the computation phase, where the data is processed, and the result collection phase, where the processor sends the processed data back to the source. 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. This is known as *block based* system model, since each phase forms a block on the time line (see Fig. 2).

For the divisible loads under consideration, such as image and video processing, Kalman filtering, matrix conversions, etc., the computation phase usually involves simple

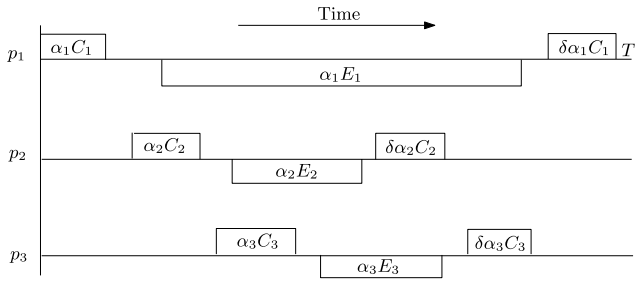

Fig. 2 A possible schedule with $m = 3$.

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 [4], [7], [20], [26], [27], [30], [31]. If the allocated load fraction is α_k, then the returned result is equal to $\delta\alpha_k$. The constant δ is application specific, and is equal for all processors for a particular load $\mathcal{J}$. In this paper, we assume $0 \leq \delta \leq 1$. The case for $\delta > 1$ is left for future research.

The time taken for computation and communication is a linearly increasing function of the size of data allocated or transferred. 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 time it takes p_k to perform the requisite processing on α_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 the computation and communication time, all results can be easily extended to affine cost models.

3.2 Problem Formulation

In the DLSRCHETS problem, the source has to partition the load $\mathcal{J}$ into fractions $\alpha_1, \ldots, \alpha_m$, and manage the allocation of these fractions to, and collection of the results from the processors $p_1, \ldots, p_m$ in the minimum possible time. Let $\mathcal{T} = \{1, \ldots, m\}$ be the set of *tasks* corresponding to the m fractions that are allocated to, and $\mathcal{R} = \{1, \ldots, m\}$ be the set of *results* that are collected from the processors $p_1, \ldots, p_m$ respectively.

Though the load fractions (tasks) can be processed independently of each other on the respective processors, the single-port communication model implicitly induces a *precedence order* on the distribution of the tasks and collection of the results. Let $\prec_a$ and $\prec_c$ be *total orders* on the sets $\mathcal{T}$ and $\mathcal{R}$ respectively, such that $\prec_a$ represents the sequence (order) in which processors are allocated tasks, and $\prec_c$ is the sequence in which results are collected from the processors at the source. Then, $i \prec_a j$ implies that task i *precedes* task j (or equivalently task j *succeeds* task i) in the allocation sequence $\prec_a$, and $i \prec_c j$ signifies that result i precedes result j in the collection sequence $\prec_c$. If $\{k \in \mathcal{T} : i \prec_a k \prec_a j\} = \emptyset$, then task i is the *immediate predecessor* of task j in $\prec_a$, and is denoted as $i \preccurlyeq_a j$. Similarly, if $\{k \in \mathcal{R} : i \prec_c k \prec_c j\} = \emptyset$, then result j is the *immediate successor* of result i in $\prec_c$, and is denoted as $i \preccurlyeq_c j$. Define $B^i_{\prec_a} := \{j \in \mathcal{T} : j \prec_a i\} \cup \{i\}$ and $F^i_{\prec_a} := \{j \in \mathcal{T} : i \prec_a j\} \cup \{i\}$, i.e., $B^i_{\prec_a}$ is the set of task i and the tasks *before* i (*predecessors* of i) in $\prec_a$, while $F^i_{\prec_a}$ is the set of task i and the *followers* (*successors*) of task i in $\prec_a$. $B^i_{\prec_c}$ and $F^i_{\prec_c}$ are defined accordingly for $\prec_c$. The *minimal element* of $\prec_a$ is defined as $\prec_a^+ := \exists!\, i \in \mathcal{T} : B^i_{\prec_a} = \{i\}$ and the *maximal element* of $\prec_a$ is defined as, $\prec_a^- := \exists!\, i \in \mathcal{T} : F^i_{\prec_a} = \{i\}$, i.e., $\prec_a^+$ and $\prec_a^-$ are the first and last tasks allocated in $\prec_a$. $\prec_c^+$ and $\prec_c^-$ are similarly defined as the first and last results returned in $\prec_c$.

For a given load $\mathcal{J}$, the objective is to minimize the total processing time T, which is defined as the time taken from the point when the source first initiates the allocation of tasks, to the point when the source completes reception of all the results. From the system model in Sect. 3.1, there are two important constraints to consider while scheduling the tasks on the processors, viz. the exclusivity of the communication medium (single-port model), and the non-overlap of communication and computation.

The *schedule* $\mathcal{S}$ of DLSRCHETS for a given load distribution α, is a pair (t, r), where, $t : \mathcal{T} \mapsto \mathbb{R}^+$ is the task allocation start time function, and $r : \mathcal{R} \mapsto \mathbb{R}^+$ is the result collection start time function. In a *feasible* schedule, the start times in t and r must satisfy the following constraints:

$$t_j - t_i \geq \alpha_i C_i \qquad \forall\, i \in \{1, \ldots, m\}, i \preccurlyeq_a j \qquad (1)$$

$$t_i \geq \sum_{j \in B^i_{\prec_a} \setminus \{i\}} \alpha_j C_j \qquad \forall\, i \in \{1, \ldots, m\} \qquad (2)$$

$$r_j - r_i \geq \delta\alpha_i C_i \qquad \forall\, i \in \{1, \ldots, m\}, i \preccurlyeq_c j \qquad (3)$$

$$T - r_i \geq \sum_{j \in F^i_{\prec_c}} \delta\alpha_j C_j \qquad \forall\, i \in \{1, \ldots, m\} \qquad (4)$$

$$r_i - t_i \geq \alpha_i C_i + \alpha_i E_i \qquad \forall\, i \in \{1, \ldots, m\} \qquad (5)$$

$$t_i \neq r_j \qquad \forall\, i, j \in \{1, \ldots, m\} \qquad (6)$$

$$r_j - t_i \geq \alpha_i C_i \qquad \forall\, j \in \{1, \ldots, m\}, \forall\, t_i < r_j \qquad (7)$$

$$t_i - r_j \geq \delta\alpha_j C_j \qquad \forall\, i \in \{1, \ldots, m\}, \forall\, r_j < t_i \qquad (8)$$

$$t_i, r_j \geq 0 \qquad \forall\, i, j \in \{1, \ldots, m\} \qquad (9)$$

The precedence constraints of $\prec_a$ are enforced by (1) and (2), while inequalities (3) and (4) impose the precedence constraints of $\prec_c$ and define the processing time T. The fact that the result collection cannot begin before the execution of the entire load fraction is complete is shown by (5). Constraints (6), (7), and (8) impose the single-port model so that no allocation and collection phase can overlap. The non-negativity of the start times is ensured by (9).

Figure 2 shows the timing diagram for a feasible schedule with $m = 3$. The time spent in communication with the source p_0 is shown above the horizontal axes, and time spent in computation by the individual processors below the horizontal axes. Since p_0 does not retain any part of the load for itself, there is no p_0 axis.

In a LIFO or FIFO schedule, the order of distribution and collection of fractions is predefined, which explicitly determines t and r once α is known. However, in the general case this is not so, and to efficiently find optimal schedules, it is

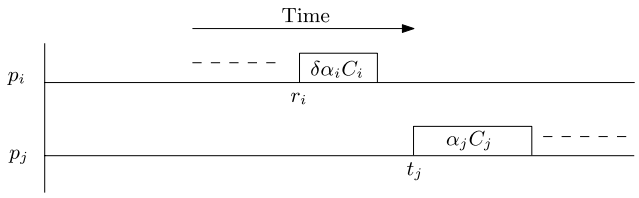

Fig. 3 Interleaved result collection.

necessary to constrain the number of possible values that t and r can take. A lemma is stated based on the following condition that reduces the range of optimal solutions to a finite number.

Condition 1 (Allocation Precedence Condition): The source should first allocate the entire load to the processors before receiving any results from the processors.

Lemma 1 (Allocation Precedence Lemma): There exists an optimal schedule for DLSRCHETS that satisfies the allocation precedence condition. (There may exist other optimal schedules that do not satisfy the allocation precedence condition.)

Proof. Consider a feasible schedule with processing time T, that satisfies (1) to (9) for a load distribution α, and an arbitrary order of allocation and collection $\prec_a$ and $\prec_c$, such that some results are collected before the load is completely allocated first.

Then, there exists at least one pair (i, j) with $i \prec_a j$, such that the result collection starting at r_i is followed by a task allocation at t_j, without any other intermediate communication phase as shown in Fig. 3.

Suppose that all load fractions in α, and all other start times in t and r are maintained the same, and only the order of collection of result i and allocation of task j is exchanged, such that the new allocation start time of task j is $t'_j = r_i$, and the new collection start time of result i is $r'_i = r_i + \alpha_j C_j$.

Since the above exchange does not alter the order of allocation of different tasks, the precedence constraints of $\prec_a$ defined by (1) and (2) still hold. Similarly, the precedence constraints of $\prec_c$, imposed by (3) and (4) also hold after the exchange. The constraints (6), (7), and (8) are valid after the exchange because the single-port model is not violated by the exchange.

Only the conditions expressed by (5) require verification. Before the exchange, the conditions $r_i - t_i \geq \alpha_i C_i + \alpha_i E_i$ and $r_j - t_j \geq \alpha_j C_j + \alpha_j E_j$ are satisfied. After the exchange, the constraints (5) are still valid because $r'_i - t_i = r_i + \alpha_j C_j - t_i > r_i - t_i$, and $r_j - t'_j = r_j - r_i > r_j - t_j$.

From the above observations, it is clear that after the reordering, all conditions for feasibility are still satisfied. Moreover, the orders $\prec_a$ and $\prec_c$ are unchanged, and no additional processing time is required for the reordering.

If a similar reordering is carried out for all such pairs (i, j), then the allocation precedence condition is satisfied with no addition in total processing time T.

Now if there is an optimal schedule for DLSRCHETS that does not satisfy the allocation precedence condition, then a reordering can be performed as mentioned above so that the schedule satisfies the allocation precedence condition without an increase in the total processing time. That is, there always exists an optimal schedule that satisfies the allocation precedence condition, and only such schedules need be considered in the search for the optimal schedule. ■

Two other basic lemma are stated before the DLSRCHETS problem is formally defined.

Lemma 2: There exists an optimal schedule for DLSRCHETS that has no idle time between any two consecutive allocation phases and any two consecutive result collection phases. (There may exist other optimal schedules that do not satisfy this condition.)

Proof. Assume that a feasible schedule that obeys (1) to (9), and in addition also satisfies the allocation precedence condition, has idle time between the consecutive communication phases (see Fig. 2). Let the processing time be T, the load distribution be α, and $(\prec_a, \prec_c)$ be the orders of allocation and collection.

According to the assumptions in the system model, all processors are available continuously and exclusively during the entire execution process, and the source can only communicate with one processor at a time. For any $i \preccurlyeq_a j$, when processor p_i completes the reception of its allocated task at time $t_i + \alpha_i C_i$, processor p_j is already available and can start receiving data immediately at $t_j = t_i + \alpha_i C_i$. Because the schedule satisfies the allocation precedence condition, load is first distributed to all the processors sequentially before result collection begins. Thus the start time of each task $i \in \mathcal{T}$ can be brought forward so that $t_i = t_{\prec_a^+} + \sum_{j \in B^i_{\prec_a} \setminus \{i\}} \alpha_j C_j$, and the inequalities (1) and (2) are reduced to equalities without exceeding T.

Following a similar logic to the one above, the result collection of each result $i \in \mathcal{R}$ can be delayed to the extent necessary to make the result collection start time $r_i = T - \sum_{j \in F^i_{\prec_c}} \delta\alpha_j C_j$, with inequalities (3) and (4) reduced to equalities and no extra time added to T.

Since any feasible schedule can be reordered in this manner to eliminate the idle time between communication phases, it follows that the optimal schedule to DLSRCHETS also has no idle time between any two consecutive allocation and result collection phases. ■

Lemma 3: There exists an optimal schedule for DLSRCHETS that has no idle time between the allocation and computation phases of each processor. (There may exist other optimal schedules that do not satisfy this condition.)

Proof. Following an argument similar to the one used in Lemma 2, since all processors are always available, they can begin computing immediately upon receiving their load fractions in the allocation phase without affecting the schedule.

Thus, any processor p_i begins computing its allocated

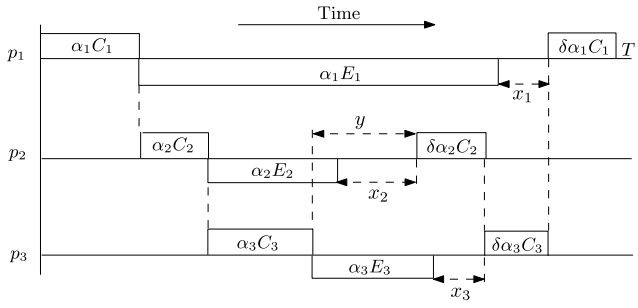

Fig. 4 Schedule with some idle time eliminated for $m = 3$.

task at time $t_{\prec_a^+} + \sum_{j \in B^i_{\prec_a}} \alpha_j C_j$ without crossing the time interval T. Since any feasible schedule can be reordered in this manner, the optimal schedule to DLSRCHETS too has no idle time between the allocation and computation phases of each processor. ■

Theorem 1: There exists an optimal schedule for DLSRCHETS that satisfies Lemmas 1 to 3.

Proof. If there exists any optimal schedule that does not satisfy any of the Lemmas 1 to 3, it can always be reordered as explained in the respective proofs to satisfy the same. ■

From Theorem 1, it follows that only those schedules that satisfy Lemmas 1 to 3 need be considered in the search for the optimal solution to DLSRCHETS. A possible timing diagram for such a schedule is shown in Fig. 4.

From the preceding discussion, it can be concluded that the start times t and r in the optimal schedule for DLSRCHETS can be determined from the sequences $\prec_a$ and $\prec_c$, and the load distribution α that minimize the processing time T. Hence instead of finding t and r as in traditional scheduling practice, the DLSRCHETS problem is formulated as a linear programming problem, to find $\prec_a$, $\prec_c$, and α that minimize T. Once the optimal values of these variables are known, it is trivial to find the optimal schedule.

The constraints (1) to (9) and the Allocation Precedence Condition are combined into a unified form, and for each processor p_i, constraints on T are written in terms of $B^i_{\prec_a}$ and $F^i_{\prec_c}$. The DLSRCHETS problem is defined as a linear program as follows.

DLSRCHETS (DLS WITH RESULT COLLECTION ON HETEROGENEOUS SYSTEMS)

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

MINIMIZE $\zeta = T$
SUBJECT TO:

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j + \alpha_k E_k + \sum_{j \in 6F^k_{\prec_c}} \delta\alpha_j C_j \leq T \qquad k = 1, \ldots, m \tag{10}$$

$$\sum_{j=1}^{m} \alpha_j C_j + \sum_{j=1}^{m} \delta\alpha_j C_j \leq T \tag{11}$$

$$\sum_{j=1}^{m} \alpha_j = \mathcal{J} \tag{12}$$

$$T \geq 0, \quad \alpha_k \geq 0 \qquad k = 1, \ldots, m \tag{13}$$

In the above formulation, for a triple $(\prec_a, \prec_c, \alpha)$, the LHS (Left Hand Side) of constraint (10) 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 (11) 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 imposed by (12). This is the *normalization equation*. The non-negativity of the decision variables is ensured by constraint (13).

The DLSRCHETS problem is defined similar to the problem addressed in [26], [30], [31]. The *throughput maximization* problem addressed in [26], [30], [31] and the *execution time minimization* problem addressed in this paper are *duals* of each other, and can be transformed from one form into the other. Because all equations are linear in the decision variables, an optimal solution to one form is also an optimal solution to the other form. However, the problem formulation given in [26], [30], [31] is applicable only for a single pair of allocation and collection sequences. LIFO and FIFO were selected as two instances of the problem and respective optimality results were derived. The formulation in this paper is completely general and the scope of the problem is global, i.e. for all possible allocation and collection sequences. The optimality results for LIFO and FIFO presented in [26], [30], [31] can be easily derived as subsets of this generic formulation.

To keep the DLSRCHETS formulation as general as possible, we have not included the idle times in the definition of the problem as in [30], [31]. In [30], [31], it is assumed in the system model itself that idle time always lies between the computation and result collection phase of a processor, when it may not always be so. The idle time can lie anywhere on the time-line. Lemmas 2 and 3 prove that idle time can be transferred to lie between computation and result collection phase of a processor.

Moreover, there is a discrepancy in the formulation used in [30], [31] because the constraints (2a) (corresponding to (10) here) are expressed as inequalities. These must actually be equalities since the idle times (x_i) are already considered in the equations.

The decision version of DLSRCHETS used to analyze the problem complexity is:

DLSRCHETS (DECISION)

INSTANCE: Heterogeneous network $\mathcal{H} = (\mathcal{P}, \mathcal{L}, \mathcal{E}, C)$, divisible load $\mathcal{J}$, time interval T.
QUESTION: Can load $\mathcal{J}$ be processed on $\mathcal{H}$, in at most T units of time?

Finding an optimal solution to the DLSRCHETS problem is surprisingly difficult. In fact, there is no known polynomial-time algorithm to find the optimal schedule for the general case considered in this paper, nor has the **NP**-completeness of DLSRCHETS been proved. The problem is in **NP**, since the values of the two permutations and the load distribution can be guessed, and it can be checked if the answer to the decision question is true or false.

4. Analysis of Optimal Solution

The processors allocated non-zero load fractions are called *participating processors* or *participants*.

Theorem 2 (Idle Time Theorem): In the optimal solution to the DLSRCHETS problem, irrespective of whether load is allocated to all available processors, at the most one of the participating processors has idle time, and the idle time exists only when the result collection begins immediately after the completion of load distribution.

Proof. For a pair $(\prec_a, \prec_c)$, the DLSRCHETS problem defined by (10) to (13) always has a feasible solution. This is because, for any load distribution α that satisfies (12), T can be made arbitrarily large to satisfy the inequalities (10) and (11). It implies that the polyhedron formed by the constraints of the DLSRCHETS problem, $P := \{\mathbf{x} \in \mathbb{R}^{m+1} : A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq 0\} \neq \emptyset$.

According to the theory of linear programming, the optimal solution to DLSRCHETS is obtained at some vertex of this polyhedron [37], [38]. As the DLSRCHETS problem has $m + 1$ decision variables and $2m + 3$ constraints, in a *non-degenerate* optimal solution, at the optimal vertex, $m + 1$ constraints out of these must be *tight*, i.e., satisfied with equality. In a *degenerate* optimal solution, more than $m + 1$ constraints are tight.

It is clear that in an optimal solution, (12) will always be tight, and T will always be greater than zero. This means that m constraints out of the remaining $2m + 1$ constraints will be tight in a non-degenerate optimal solution. There are two possible ways to proceed with the analysis at this point depending on the assumption regarding the allocated load fractions in the optimal solution.

1. $\forall\ k \in \{1, \ldots, m\} : \alpha_k > 0$.
   In this case, all the load fractions are assumed to be always greater than zero, i.e. number of participants is m. Since all decision variables are positive, there can be no degeneracy [38, Chapter 3].

   It leaves only $m + 1$ constraints (10) and (11), out of which m will be tight in the optimal solution. Hence, in the optimal solution, either,

   (a) the m constraints (10) are tight, and the (11) constraint is not, or
   (b) the (11) constraint is tight and one of the (10) constraints is not.

   If any constraint from (10) and (11) is not tight in the optimal solution, it implies a *shortfall* in the LHS as compared to the optimal processing time. In constraints (10) this shortfall represents idle time in a processor, while in (11) it represents the intervening time interval between completion of load distribution from the source and the start of result transfer to the source.

   Thus, if the option (a) above is true, then none of the processors have any idle time in the optimal solution. If the option (b) is true, then one of the processors has idle time, and since this happens only when constraint (11) is tight, it means that idle time in a processor exists only when result transfer to the source begins immediately after completion of load allocation is completed. This is similar to the analysis in [30], [31].
2. $\exists\ k \in \{1, \ldots, m\} : \alpha_k = 0$.
   In this case, some of the processors can be allocated zero load in the optimal solution.

   The analysis has two parts — for non-degenerate and degenerate optimal solutions.

   Non-degenerate Optimal Solution

   If there are p ($p \leq m$) participants in the optimal solution, then $m - p$ constraints of (13) are necessarily tight. This means that out of the $m + 1$ constraints (10) and (11), only p constraints will be tight in the optimal solution. Hence, in the optimal solution, either,

   (a) p of the (10) constraints are tight, $m - p$ of the (10) constraints are not tight, and the (11) constraint is not tight, or
   (b) the (11) constraint is tight, $p - 1$ of the (10) constraints are tight, and $m - p + 1$ of the (10) constraints are not tight.

   In the optimal solution, if the option (a) is true, then $m - p$ processors have idle time, while if the option (b) is true, then $m - p + 1$ processors have idle time.

   Since $m - p$ processors are not allocated load, it is obvious that they are idle throughout in either of the above two options. The additional processor with idle time if the option (b) is true has to be one of the participating processors. This means that idle time in a participating processor exists only when the result collection begins immediately upon completion of load allocation.

   Degenerate Optimal Solution

   Similar to the non-degenerate case, if there are p ($p \leq m$) participants in the optimal solution, then $m - p$ constraints of (13) are necessarily tight. Since the optimal solution is degenerate, *more than* p constraints out of the $m + 1$ constraints (10) and (11) will be tight.

This means that in the optimal solution, irrespective of whether the (11) constraint is tight, *at least* p of the (10) constraints are tight, and *less than* $m - p$ of the (10) constraints are not tight. Since $m - p$ processors are necessarily idle, some of the (10) constraints corresponding to the processors allocated zero load are tight in the degenerate solution.

Since $\forall\, k \in \{1, \ldots, m\},\ B^k_{\prec_a}, F^k_{\prec_c} \subseteq \{1, \ldots, m\}$, it implies that,

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j \leq \sum_{j=1}^{m} \alpha_j C_j \qquad k \in \{1, \ldots, m\}$$

and

$$\sum_{j \in F^k_{\prec_c}} \delta\alpha_j C_j \leq \sum_{j=1}^{m} \delta\alpha_j C_j \qquad k \in \{1, \ldots, m\}$$

It follows that,

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j + \sum_{j \in F^k_{\prec_c}} \delta\alpha_j C_j \leq \sum_{j=1}^{m} \alpha_j C_j + \sum_{j=1}^{m} \delta\alpha_j C_j \qquad k \in \{1, \ldots, m\} \quad (14)$$

If (11) is not tight, then the RHS (Right Hand Side) of (14) is strictly less than T. That is,

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j + \sum_{j \in F^k_{\prec_c}} \delta\alpha_j C_j < T \quad k \in \{1, \ldots, m\} \quad (15)$$

If $\exists\, k \in \{1, \ldots, m\} : \alpha_k = 0$, then $\alpha_k E_k = 0$, and from (15), it immediately follows that the corresponding constraint from (10) can never be tight.
Thus, a constraint corresponding to a processor p_k allocated zero load is tight in the optimal solution only if

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j + \sum_{j \in F^k_{\prec_c}} \delta\alpha_j C_j - T = 0 \quad (16)$$

or equivalently if (14) is satisfied with an equality, *and* the RHS of (14) is equal to T, i.e, the (11) constraint is tight.

It is now clear that a degenerate optimal solution exists only when the (11) constraint is tight, and the condition (16) is satisfied. To find when the condition is satisfied, consider the case where for some pair $(\prec_a, \prec_c)$, one or more of the processors allocated zero load follow each other at the end of the allocation sequence and the start of the result collection sequence in the optimal solution.

For example, if $\alpha_i, \alpha_j, \alpha_k = 0$, and one or more of the following occur (the list is not exhaustive):

- $\prec_a^- = i$ and $\prec_c^+ = i$
- $i \preccurlyeq_a j$, $\prec_a^- = j$ and $\prec_c^+ = i$
- $i \preccurlyeq_a j$, $\prec_a^- = j$, $\prec_c^+ = k$ and $k \preccurlyeq_c i$

Only if such tail-end zero-load processors exist, then (14) is satisfied with an equality. Finally, if constraint (11) is tight in the optimal solution, then it follows that the constraints corresponding to these processors are tight.

The linear program obtained after eliminating the redundant constraints corresponding to the tail-end zero-load processors has a non-degenerate optimal solution. This is because, the feasible region defined by the constraints of the non-degenerate problem does not change after addition of the redundant constraints. Hence only a single participant processor has idle time in the degenerate optimal solution.

From the preceding discussion on the optimal solution to the linear program for a pair $(\prec_a, \prec_c)$, it follows that in the optimal solution to the DLSRCHETS problem, $(\prec_a^*, \prec_c^*, \alpha^*)$, at the most one participating processor can have idle time. The idle time occurs *only when* the result collection from processor $\prec_c^+$ starts immediately after completion of load allocation to processor $\prec_a^-$. ■

There are $m!$ possible permutations each of $\prec_a$ and $\prec_c$, and the linear program has to be evaluated $(m!)^2$ times to determine the globally optimum solution $(\prec_a^*, \prec_c^*, \alpha^*)$ for DLSRCHETS. Since the solution to the linear program is completely determined by the values of δ, C and $\mathcal{E}$, along with the pair $(\prec_a, \prec_c)$, it is not possible at this stage to predict which of the processors or how many processors will be allocated zero load.

If the load distribution is constrained to be strictly positive, i.e., $\forall \alpha_k > 0 \Rightarrow p = m$, then there is only one processor with idle time in the optimal solution when (11) is tight. As proved in [30], [31], for the FIFO schedule, this processor can always be chosen to be processor $\prec_a^-$. However, this may not be true in general. For a LIFO schedule under the same constraints, no processor ever has idle time in the optimal solution because (11) can never be tight.

5. Conclusion

In this paper, the DLSRCHETS problem for the scheduling of divisible loads on heterogeneous systems and considering the result collection phase was formulated and analyzed.

Among the main contributions of this paper are the generic formulation of the DLSRCHETS problem and a proof of the allocation precedence condition. The optimal solution to DLSRCHETS was analyzed in detail and it was proved that irrespective of whether load is allocated to all processors or not, at the most one processor that is allocated load has idle time. In the general case it is not possible to predict which participating processor is the one with idle time. It was proved that the idle time exists only when the source starts to receive results immediately after completion of the distribution of the load to the participating processors.

As there is no known polynomial time algorithm to obtain the optimal solution, future work involves proposing

heuristic algorithms to solve the DLSRCHETS problem.

Acknowledgements

The authors would like to sincerely thank the reviewers for their insightful comments on the previous drafts of this paper.

References

[1] D.P. Anderson, E. Korpela, and R. Walton, "High-performance task distribution for volunteer computing," First IEEE Intl. Conf. on e-Science and Grid Technologies, Melbourne, Dec. 2005.
[2] D.P. Anderson and G. Fedak, "The computational and storage potential of volunteer computing," IEEE/ACM International Symposium on Cluster Computing and the Grid, Singapore, May 2006.
[3] D.P. Anderson and J. McLeod VII, "Local scheduling for volunteer computing," Workshop on Large-Scale, Volatile Desktop Grids (PC-Grid 2007) held in conjunction with the IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS), Long Beach, CA, March 2007.
[4] 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.
[5] V. Bharadwaj, D. Ghose, and T.G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," Cluster Computing, vol.6, no.1, pp.7–17, Jan. 2003.
[6] T. Robertazzi, http://www.ece.sunysb.edu/~tom/dlt.html, April 2005.
[7] N. Comino and V.L. Narasimhan, "A novel data distribution technique for host-client type parallel applications," IEEE Trans. Parallel Distrib. Syst., vol.13, no.2, pp.97–110, Feb. 2002.
[8] 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 Vis. Comput., vol.18, no.1, pp.919–938, Jan. 2000.
[9] 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., vol.5, no.9, pp.968–976, Sept. 1994.
[10] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-installment load distribution in tree networks with delays," IEEE Trans. Aerosp. Electron. Syst., vol.31, no.2, pp.555–567, April 1995.
[11] 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., vol.11, no.12, pp.1288–1305, Dec. 2000.
[12] X. Li, V. Bharadwaj, and C.C. Ko, "Divisible load scheduling on single-level tree networks with buffer constraints," IEEE Trans. Aerosp. Electron. Syst., vol.36, no.4, pp.1298–1308, Oct. 2000.
[13] 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., vol.43, no.10, pp.1184–1196, Oct. 1994.
[14] S. Bataineh and B. Al-Asir, "An efficient scheduling algorithm for divisible and indivisible tasks in loosely coupled multiprocessor systems," Software Engineering Journal, vol.9, no.1, pp.13–18, Jan. 1994.
[15] S. Bataineh and T.G. Robertazzi, "Performance limits for processors with divisible jobs," IEEE Trans. Aerosp. Electron. Syst., vol.33, no.4, pp.1189–1198, Oct. 1997.
[16] J. Sohn and T.G. Robertazzi, "Optimal divisible job load sharing for bus networks," IEEE Trans. Aerosp. Electron. Syst., vol.32, no.1, pp.34–40, Jan. 1996.
[17] J. Sohn, T.G. Robertazzi, and S. Luryi, "Optimizing computing costs using divisible load analysis," IEEE Trans. Parallel Distrib. Syst., vol.9, no.3, pp.225–234, March 1998.
[18] T.G. Robertazzi, "Processor equivalence for daisy chain load sharing processors," IEEE Trans. Aerosp. Electron. Syst., vol.29, no.4, pp.1216–1221, Oct. 1993.
[19] Y.C. Cheng and T.G. Robertazzi, "Distributed computation for a tree network with communication delays," IEEE Trans. Aerosp. Electron. Syst., vol.26, no.3, pp.511–516, May 1990.
[20] 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., vol.9, no.5, pp.429–441, May 1998.
[21] V. Mani and D. Ghose, "Distributed computation in linear networks: Closed-form solutions," IEEE Trans. Aerosp. Electron. Syst., vol.30, no.2, pp.471–483, April 1994.
[22] D. Ghose and V. Mani, "Distributed computation with communication delays: Asymptotic performance analysis," J. Parallel Distrib. Comput., vol.23, no.3, pp.293–305, Dec. 1994.
[23] D. Ghose and H.J. Kim, "Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays," J. Parallel Distrib. Comput., vol.55, no.1, pp.32–59, Nov. 1998.
[24] H.J. Kim, G. in Jee, and J.G. Lee, "Optimal load distribution for tree network processors," IEEE Trans. Aerosp. Electron. Syst., vol.32, no.2, pp.607–612, April 1996.
[25] C.H. Lee and K.G. Shin, "Optimal task assignment in homogeneous networks," IEEE Trans. Parallel Distrib. Syst., vol.8, no.2, pp.119–129, Feb. 1997.
[26] O. Beaumont, L. Marchal, and Y. Robert, "Scheduling divisible loads with return messages on heterogeneous master-worker platforms," Research Report 2005-21, May 2005.
[27] A. Rosenberg, "Sharing partitionable workload in heterogeneous NOWs: Greedier is not better," IEEE International Conference on Cluster Computing, Newport Beach, CA, pp.124–131, Oct. 2001.
[28] 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., vol.15, no.4, pp.1–12, April 2004.
[29] O. Beaumont, A. Legrand, and Y. Robert, "Optimal algorithms for scheduling divisible loads on heterogeneous systems," International Parallel and Distributed Processing Symposium, (IPDPS) 2003, p.14, April 2003.
[30] 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.
[31] O. Beaumont, L. Marchal, V. Rehn, and Y. Robert, "FIFO scheduling of divisible loads with return messages under the one port model," Proc. Heterogeneous Computing Workshop HCW'06, April 2006.
[32] D. Yu and T.G. Robertazzi, "Scalable scheduling in parallel processors," Proc. Conference on Information Sciences and Systems, Princeton, NJ, March 2002.
[33] J.T. Hung and T.G. Robertazzi, "Scalable scheduling for clusters and grids using cut through switching," Intl. J. of Computers and their Applications, vol.26, no.3, pp.147–156, 2004.
[34] D. Yu and T.G. Robertazzi, "Divisible load scheduling for grid computing," Proc. International Conference on Parallel and Distributed Computing Systems (PDCS 2003), Los Angeles, CA, USA, Nov. 2003.
[35] D.A.L. Piriyakumar and C.S.R. Murthy, "Distributed computation for a hypercube network of sensor-driven processors with communication delays including setup time," IEEE Trans. Syst., Man, Cybern. A, Syst. Humans, vol.28, no.3, pp.245–251, March 1998.
[36] R. Thakur and W. Gropp, Open Issues in MPI Implementation, Lecture Notes in Computer Science, vol.4697, Springer Berlin/Heidelberg, 2007.
[37] G.B. Dantzig, Linear Programming and Extensions, Princeton Univ. Press, Princeton, NJ, 1963.
[38] R.J. Vanderbei, Linear Programming: Foundations and Extensions,

2nd ed., International Series in Operations Research & Management, vol.37, Kluwer Academic Publishers, 2001.

Abhay Ghatpande received his B.E. 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. In 1997 he started his career in Larsen & Toubro Ltd., as software engineer. In 2000, he helped set up MoTech Software's Japan branch office. In 2002 he turned to research. 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.

Hidenori Nakazato received his B. Engineering degree in Electronics and Telecommunications from Waseda University in 1982, and his MS and Ph.D. degrees in Computer Science from University of Illinois in 1989 and 1993, respectively. He was with Oki Electric from 1982 to 2000 where he developed equipment for public telephony switches, distributed environment for telecommunications systems, and communications quality control mechanisms. He has been a Professor at Graduate School of Global Information and Telecommunications Studies, Waseda University since 2000. His research interests include performance issues in distributed systems and networks, cooperation mechanisms of distributed programs, distributed real-time systems, and network QoS control.

Olivier Beaumont received his Ph.D. from the University of Rennes in 1999. From 1999 to 2001, he was assistant professor at Ecole Normale Supérieure de Lyon. He was appointed at ENSEIRB in Bordeaux. In 2004, he defended his "habilitation à 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.

Hiroshi Watanabe received the B.E., M.E. and Ph.D. degrees from Hokkaido University, Japan, in 1980, 1982 and 1985, respectively. He joined Nippon Telegraph and Telephone Corporation (NTT) in 1985, and engaged in R&D for image and video coding systems at NTT Human Interface Labs. (Later NTT Cyber Space Labs.) until August 2000. He has also engaged in developing JPEG, MPEG standards under JTC 1/SC 29. From Sept. 2000, he is a Professor at Graduate School of Global Information and Telecommunication Studies. He has been ISO/IEC JTC 1/SC 29 Chairman since November 1999. He is an associate editor of IEEE Transactions of Circuits and Systems for Video Technology. He is a member of a steering committee of PCSJ, IEEE, IPSJ, ITE, IIEEJ.