SPORT: An Algorithm for Divisible Load Scheduling with Result Collection on Heterogeneous Systems

Abhay GHATPANDE†, Student Member, Hidenori NAKAZATO†, Member, Olivier BEAUMONT††, Nonmember, and Hiroshi WATANABE†, Member

SUMMARY Divisible Load Theory (DLT) is an established mathematical framework to study Divisible Load Scheduling (DLS). However, traditional DLT does not address the scheduling of results back to source (i.e., result collection), nor does it comprehensively deal with system heterogeneity. 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 fifo (First In, First Out) and lifo (First In, Last Out) type of schedules as solutions to dlsrchets. In this paper, a new polynomial time heuristic algorithm, sport (System Parameters based Optimized Result Transfer), is proposed as a solution to the dlsrchets problem. With the help of simulations, it is proved that the performance of sport is significantly better than existing algorithms. The other major contributions of this paper include, for the first time ever, (a) the derivation of the condition to identify the presence of idle time in a fifo schedule for two processors, (b) the identification of the limiting condition for the optimality of lifo and lifo schedules for two processors, and (c) the introduction of the concept of equivalent processor in DLS for heterogeneous systems with result collection.

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

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 data-set processing, image processing, signal processing, computation of Hough transforms, database search, simulations, and matrix computations. Divisible Load Theory (DLT) is the mathematical framework that has been established to study Divisible Load Scheduling (DLS) [1]–[22].

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 [17]. 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.

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 floating point or boolean value. This is not that uncommon, but the other cases need due consideration. All papers that have addressed result collection to date, have advocated simplistic fifo (First In, First Out) and lifo (First In, Last Out) sequences or variants thereof as solutions [1], [16], [17], [23]–[26]. It has been proved in [23] that lifo and lifo are not always optimal, and as this paper shows, the performance of these algorithms varies widely depending on whether it is the network links that are heterogeneous, or the processor speeds, or both.

Several papers have dealt with DLS on heterogeneous systems to date [1], [4], [23]–[26]. 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, [24], [25] proved that the optimal lifo 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 establishment of the theoretical base for the sport (System Parameters based Optimized Result Transfer) algorithm, and the proof of its performance through simulation. This paper includes for the first time ever, (a) the derivation of the condition to identify the presence of idle time in a lifo schedule for two processors, (b) the identification of the limiting condition for the optimality of lifo and lifo schedules for two processors, and (c) the introduction of the concept of an equivalent processor in DLS for heterogeneous systems with result collection.

---

© 2008 The Institute of Electronics, Information and Communication Engineers
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 \( l_k \), and \( E_k, C_k \geq 0 \), \( E_k, C_k \) not simultaneously zero, for \( k \in \{1, \ldots, m\} \).

The values in \( \mathcal{E} \) and \( \mathcal{C} \) are assumed to be 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 communication 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 course of execution of the entire process and have sufficient buffer capacity to receive the entire load fraction in a single installment from the source. 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, 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 a block based system model, since each phase forms a block on the time line (see Fig. 2). The source starts receiving results from the child processors only after the entire load is distributed to the child processors 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, [4],[17],[23]–[26]. If the allocated load fraction is \( \alpha_k \), then the returned result is equal to \( \delta \alpha_k \), where \( 0 \leq \delta \leq 1 \). The constant \( \delta \) is application specific, and is the same for all processors for a particular load \( \mathcal{J} \). For a load fraction \( \alpha_k \), \( \alpha_k C_k \) 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 \( \alpha_k \), and \( \delta \alpha_k C_k \) is the time it takes \( p_k \) to transmit the results back to \( p_0 \).

Let \( \sigma_a \) and \( \sigma_c \) be 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 \( k \in \{1, \ldots, m\} \). Let \( \sigma_a(l) \) and \( \sigma_c(l) \) be two lookup functions that return the index of the processor \( l \in \{1, \ldots, m\} \) in the allocation and collection sequences respectively.

The divisible load problem is defined as follows:
Given a heterogeneous network $\mathcal{H} = (\mathcal{P}, \mathcal{L}, \mathcal{E}, C)$, and a divisible load $J$, find the sequence pair $(\sigma_a, \sigma_e)$, and load distribution $\alpha = [\alpha_1, \ldots, \alpha_m]$ that

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

Subject To:

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

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

$$T \geq 0, \quad \alpha_k \geq 0 \quad k = 1, \ldots, m$$

In the above formulation, for a pair $(\sigma_a, \sigma_e)$, (1) imposes the no-overlap constraint. The single-port communication model is enforced by (2). 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 $(\sigma_a, \sigma_e)$ can be solved using standard linear programming techniques in polynomial time [27]. There are $m!$ possible permutations each of $\sigma_a$ and $\sigma_e$, and the linear program has to be evaluated $(m!)^2$ times to determine the globally optimal solution $(\sigma^*_a, \sigma^*_e, \alpha^*)$. Clearly, this is impractical to do for more than a few processors.

3. Analysis of Two Slave Processors

As mentioned in Sect. 1, traditional DLT assumes that load is always distributed to all the processors in a network, and that a processor is idle only up to the point where it starts receiving its load fraction. In the case of a heterogeneous network with result collection, this may not always be true. As shown in Fig. 2, idle time $x_i$ may potentially be present in each processor $i$ because it has to wait for another processor to release the communication medium for result transfer. In [24],[25] it has been proved that in the optimal solution to DLSRCHETS, $\forall i \in \{1 \ldots m\}$, $x_i = 0$, if and only if $y > 0$, and that there exists a unique $x_i > 0$ if and only if $y = 0$, where $y$ is the intervening time interval between the end of allocation phase of processor $\sigma_a[m]$ and the start of result collection from processor $\sigma_e[1]$. For the fifo schedule in particular, processor $\sigma_a[m]$ can always be selected to have idle time when $y = 0$, i.e., in the fifo schedule, $x_{\sigma_a[m]} > 0$ if and only if $y = 0$. In the lifo schedule, since $y > 0$ always, no processor has idle time, i.e., $\forall i \in \{1 \ldots m\}$, $x_i = 0$ always [23].

In the general case considered in this paper, for a pair $(\sigma_a, \sigma_e)$, the solution to the linear program defined by (1) to (4) is completely determined by the values of $\delta$, $E$, $C$, and it is not possible at this stage to predict which processor is the one that has idle time in the optimal solution. In fact, it is possible that not all processors are allocated load in the optimal solution (in which case some processors are idle throughout). The processors that are allocated load for computation are known as the participating processors (or participants).

Thus any heuristic algorithm for DLSRCHETS must find both — the number of participants, and the load fractions allocated to them. A polynomial time heuristic algorithm,スポーツ is proposed in this paper that does this simultaneously. The foundation of the Sports algorithm is laid first by analyzing the case of a network with two slave processors.

3.1 Optimal Schedule in Two-Slave Network

The heterogeneous network $\mathcal{H} = (\mathcal{P}, \mathcal{L}, \mathcal{E}, C)$ with $m = 2$ is shown in Fig. 3. It is defined as in Sect. 2 by $\mathcal{P} = \{p_0, p_1, p_2\}$, $\mathcal{L} = \{l_1, l_2\}$, $\mathcal{E} = \{E_1, E_2\}$, and $C = \{C_1, C_2\}$. Without loss of generality, it is assumed that $J = 1$ and $C_1 \leq C_2$. No assumptions are possible regarding the relationship between $E_1$ and $E_2$, or $C_1 + E_1 + \delta C_1$ and $C_2 + E_2 + \delta C_2$ (or equivalently $C_1 + E_1$ and $C_2 + E_2$)

An important parameter, $p_k$, known as the network parameter is introduced, which shows how fast (or slow) the processor computation parameter $E_k$ is with respect to the communication parameter $C_k$ of its network link:

$$p_k = \frac{E_k}{C_k} \quad k = 1, \ldots, m$$

The master $p_0$ distributes the load $J$ between the two slave processors $p_1$ and $p_2$ so as to minimize the processing time $T$. Depending on the values of $\delta$, $E$ and $C$, there are several possibilities:

1. Load is distributed to $p_1$ only with processing time

$$T^1 = C_1 + E_1 + \delta C_1 = C_1(1 + \delta + \rho_1)$$

2. Load is distributed to $p_2$ only with processing time

$$T^2 = C_2 + E_2 + \delta C_2 = C_2(1 + \delta + \rho_2)$$

3. Load is distributed to both $p_1$ and $p_2$. From [23]–[25], it can be proved that as long as $C_1 \leq C_2$, only the schedules shown in Figs. 4(a), 4(b), and 4(c) can be optimal.
for a two-slave network. These schedules are referred to as Schedule $f$, Schedule $l$, and Schedule $g$ respectively. Superscripts $f$, $l$, and $g$ are used to distinguish the three schedules. The processing times are given in Appendix.

A few lemmas to determine the optimal schedule for a two-slave network are now stated.

**Lemma 1:** It is always advantageous to distribute the load to both the processors, rather than execute it on the individual processors (for the system model under consideration).

**Proof.** From (A·4), (A·5), (A·16), (A·17), (A·24), and (A·25), it can be concluded that:

1. $\delta C_2 \leq C_1(1 + \delta + \rho_1) \Rightarrow T^f \leq T^l$
2. $C_1 \leq C_2 \Rightarrow T^l \leq T^2$
3. $E_1 \geq 0 \Rightarrow T^l \leq T^1$
4. $C_1 \leq C_2 \Rightarrow T^l \leq T^2$
5. $\delta C_2 \leq C_1(1 + \delta + \rho_1) \Rightarrow T^g \leq T^1$
6. $C_1 \leq C_2 \Rightarrow T^g \leq T^2$

By assumption, $C_1 \leq C_2$. Hence, from points 2, 4, and 6 above, execution time of Schedules $f$, $l$, and $g$ is always smaller than $T^2$.

By definition, $E_1 > 0$. From points 1, 3, and 5 above, as long as $\delta C_2 \leq C_1(1 + \delta + \rho_1)$, execution time of Schedules $f$, $l$, and $g$ is less than $T^1$.

Finally, if $\delta C_2 > C_1(1 + \delta + \rho_1)$, then $T^f$ and $T^g$ are greater than $T^1$, but since $T^l$ is always less than $T^1$, load can be distributed in Schedule $l$ to reduce the processing time.

Thus, it is always advantageous to distribute load to two processors instead of one under the system model under consideration.

Lemma 1 is important because it helps solve determine the number of participants in a general schedule. Details are given in Sect. 4.1.

From Lemma 1, if $\delta C_2 \leq C_1(1 + \delta + \rho_1)$, then any of the Schedules $f$, $g$, or $l$ could be optimal. The limiting condition between Schedule $f$ and Schedule $g$ is stated in the following lemma.

**Lemma 2:** $\rho_1 \rho_2 \leq \delta$ is a necessary and sufficient condition to indicate the presence of idle time in the fifo schedule (i.e. Schedule $g$).

**Proof.** If the values of $\delta$, $E$, and $C$, are such that they necessitate the presence of idle time in a fifo schedule, then the schedule can be reduced to one similar to that shown in Fig. 4(c).

In that case, idle time in processor $p_2$ occurs only when $\rho_2 E_2 \leq \delta C_1$. From (A·22), this condition reduces to $\rho_1 \rho_2 \leq \delta$.

The simplicity of the condition to detect the presence of idle time in the fifo schedule is both pleasing and surprising, and as far as can be judged, has been derived for the first time ever. Further confirmation of this condition is obtained in Sect. 3.2.

The following theorem can now be stated.

**Theorem 1** (Optimal Schedule Theorem): The optimal schedule for a two-slave network can be found as follows:

1. If $\delta C_2 > C_1(1 + \delta + \rho_1)$, then Schedule $l$ is optimal.
2. If $\delta C_2 \leq C_1(1 + \delta + \rho_1)$, and both (A·22) and (A·26) hold, then Schedule $g$ is optimal. Else if (A·26) does not hold, then Schedule $l$ is optimal.
3. If $\delta C_2 \leq C_1(1 + \delta + \rho_1)$, (A·22) does not hold, and condition (A·18) holds, then Schedule $f$ is optimal. Else if (A·18) does not hold, then Schedule $l$ is optimal.

**Proof.** The proof follows directly from Lemmas 1 and 2, and (A·18) and (A·26).

All the conditions use only the data provided in the definition of the problem, with the exception of (A·18) which requires the computation of $T^f$.

It can be argued that the optimal schedule can be determined by directly computing the values of $T^f$, $T^l$, and $T^g$ using (A·3), (A·14), and (A·23) respectively. While this fact cannot be denied, it defeats the ultimate purpose of
this research, which is to identify relationships between the system parameters that influence the optimality of different schedules.

Another very interesting insight into the problem is provided by (A-18). The value $C_1C_2/(C_2 - C_1)$ forms a limiting condition between the optimality of Schedules $f$ and $l$. It can be seen that as long as $T^f$ is smaller than this value, it is also smaller than $T^l$.

As the network links become homogeneous, the difference $(C_2 - C_1)$ becomes small, and Schedule $f$ is likely to be optimal because $T^f$ would be less than the large value $C_1C_2/(C_2 - C_1)$. Similarly, as the network links become heterogeneous, the value $C_1C_2/(C_2 - C_1)$ becomes small, and Schedule $l$ would tend to be optimal because $T^l$ can easily exceed this value.

Rosenberg reached a similar conclusion with the help of simulations [26]. However this condition is analytically derived for the first time in DLT literature.

The resulting network is called the equivalent network

$$\text{Schedule} \quad \text{distribution is} \quad \text{works for Schedules} \quad \text{works for Schedules} \quad \text{work. Similarly Figs. 5(b) and 5(c) show the equivalent network}.$$
Since by definition, $0 \leq \alpha_2 \leq 1$, it immediately follows that $C_1 \leq C_{1:2} \leq C_2$.

Similarly, from the definition of equivalent network, $E_{1:2} = \alpha_1 E_1 - \alpha_2 C_2 = \alpha_2 E_2 - \delta \alpha_1 C_1$ and $E_{1:2} = \alpha_2 E_2 = \alpha_1 E_1 - \alpha_2 C_2 (1 + \delta)$. Since $0 \leq (\alpha_1, \alpha_2) \leq 1$, it follows that $E_{1:2} = E_1, E_2$ and $E_{1:2} \leq E_1, E_2$.

Finally, $E_{1:2} = 0 \Rightarrow E_{1:2} \leq E_1, E_2$, since by definition, $E_1, E_2 > 0$.

The equivalent processor for Schedule $f$ provides additional confirmation of the condition for the presence of idle time in a FIFO schedule (i.e., use of Schedule $g$). It is known that idle time can exist in a FIFO schedule only when the intervening time interval $y = 0$. According to the definition of equivalent processor, this interval corresponds to the equivalent computation capacity $E_{1:2}$. From (8), this value becomes zero only when $\rho_1 \rho_2 - \delta = 0$. Thus, if $\rho_1 \rho_2 < \delta$, then idle time must exist in the FIFO schedule.

The equivalent processor enables replacement of two processors by a single processor with communication parameter with a value that lies between the values of communication parameters of the original two links. Because of this property, if the processors are arranged so that $C_1 \leq C_2 \leq \ldots \leq C_m$, and two processors are combined at a time sequentially, then the resultant equivalent processor does not disturb the order of the sequence. This property is exploited in the $spor$ algorithm, which is described next.

4. Proposed Algorithm

The proposed $spor$ algorithm is as follows.

Algorithm 1 (spor):

1: Arrange $p_1, \ldots, p_m$ s.t. $C_1 \leq C_2 \leq \ldots \leq C_m$
2: $\sigma_\alpha \leftarrow 1, \sigma_c \leftarrow 1, \alpha_1 \leftarrow 1$
3: for $k := 2$ to $m$
4: $C_1 \leftarrow C_{1:k-1}, E_1 \leftarrow E_{1:k-1}, C_2 \leftarrow C_k, E_2 \leftarrow E_k$
5: if $\delta C_2 > C_1 (1 + \delta + \rho_1)$ then
6: /* $T^f < T^l$, use Schedule $l$ */
7: call schedule_lifo($C_1, C_2, E_1, E_2$)
8: else
9: /* Need to check other conditions */
10: if $\rho_1 \rho_2 \leq \delta$ then
11: /* Possibility of idle time */
12: if $C_2 \leq C_1 (1 + (1 + \rho_1) \rho_2)$ then
13: /* $T^g < T^l$, use Schedule $g$ */
14: call schedule_idle($C_1, C_2, E_1, E_2$)
15: break for
16: else
17: /* $T^l < T^g$, use Schedule $l$ */
18: call schedule_lifo($C_1, C_2, E_1, E_2$)
19: end if
20: else
21: /* No idle time present */
22: if $T^f < C_1 C_2$ then
23: /* $T^f < T^l$, use Schedule $f$ */
24: call schedule_fifo($C_1, C_2, E_1, E_2$)
25: else
26: /* $T^l < T^f$, use Schedule $l$ */
27: call schedule_lifo($C_1, C_2, E_1, E_2$)
28: end if
29: end if
30: end if
31: end for

Fig. 5 Timing diagrams for equivalent processor.
GHATPANDE et al.: SPORT: AN ALGORITHM FOR DLSRCHETS

32: \( n \leftarrow \text{numberOfProcessorsUsed} \)
33: /* Find load fractions */
34: \[
\begin{align*}
\alpha_k & \leftarrow \alpha_k \cdot \prod_{j=2}^{n} \alpha_{1:j} & \text{if } k = 1 \\
\alpha_k & \leftarrow \alpha_k \cdot \prod_{j=k}^{n} \alpha_{1:j} & \text{if } k = 2, \ldots, n
\end{align*}
\]
35: \( T \leftarrow C_{1:n} + E_{1:n} + \delta \cdot C_{1:n} \)

**procedure** schedule_idle\( (C_1, C_2, E_1, E_2) \)

1: \( \alpha_{1:k-1} \leftarrow \frac{C_2}{C_1 \rho_1 + C_2} \)
2: \( \alpha_k \leftarrow \frac{C_2 r_f^l}{C_1 r_f^l + C_2 r_f^2} \)
3: /* Update sequences for fifo */
4: \( \sigma_n \leftarrow [\sigma_n k] \)
5: \( \sigma_c \leftarrow [\sigma_c k] \)
6: /* Compute equivalent processor parameters */
7: \( C_{1:x} \leftarrow \frac{C_1 C_2 (1 + \rho_1)}{C_1 \rho_1 + C_2} \)
8: \( E_{1:x} \leftarrow 0 \)
9: \( \text{numberOfProcessorsUsed} \leftarrow k \)
10: \( \text{return} \)

**end procedure**

**procedure** schedule_lifo\( (C_1, C_2, E_1, E_2) \)

1: \( r_f^l \leftarrow \rho_1 \)
2: \( r_f^2 \leftarrow 1 + \delta + \rho_2 \)
3: \( \alpha_{1:k-1} \leftarrow \frac{C_2 r_f^l}{C_1 r_f^l + C_2 r_f^2} \)
4: \( \alpha_k \leftarrow \frac{C_1 r_f^l + C_2 r_f^2}{C_1 r_f^l} \)
5: /* Update sequences for lifo */
6: \( \sigma_n \leftarrow [\sigma_n k] \)
7: \( \sigma_c \leftarrow [\sigma_c k] \)
8: /* Compute equivalent processor parameters */
9: \( C_{1:x} \leftarrow \frac{C_1 C_2 (r_f^l + r_f^2)}{C_1 r_f^l + C_2 r_f^2} \)
10: \( E_{1:x} \leftarrow 0 \)
11: \( \text{numberOfProcessorsUsed} \leftarrow k \)
12: \( \text{return} \)

**end procedure**

**procedure** schedule_fifo\( (C_1, C_2, E_1, E_2) \)

1: \( r_f^l \leftarrow \delta + \rho_1 \)
2: \( r_f^2 \leftarrow 1 + \rho_2 \)
3: \( \alpha_{1:k-1} \leftarrow \frac{C_2 r_f^l}{C_1 r_f^l + C_2 r_f^2} \)
4: \( \alpha_k \leftarrow \frac{C_1 r_f^l + C_2 r_f^2}{C_1 r_f^l} \)
5: /* Update sequences for fifo */
6: \( \sigma_n \leftarrow [\sigma_n k] \)
7: \( \sigma_c \leftarrow [\sigma_c k] \)
8: /* Compute equivalent processor parameters */
9: \( C_{1:x} \leftarrow \frac{C_1 C_2 (r_f^l + r_f^2)}{C_1 r_f^l + C_2 r_f^2} \)
10: \( E_{1:x} \leftarrow 0 \)
11: \( \text{numberOfProcessorsUsed} \leftarrow k \)
12: \( \text{return} \)

**end procedure**

4.1 Algorithm Explanation

If \( m = 2 \), finding the optimal schedule and load distribution is trivial. In the following, it is assumed that \( m \geq 3 \). At the start, the processors are arranged so that \( C_1 \leq C_2 \leq \ldots \leq C_m \), and two processors with the fastest communication links are selected. The optimal schedule and load distribution for the two processors are found according to Theorem 1. If Schedule \( f \) or \( l \) is found optimal, then the two processors are replaced by their equivalent processor. In either case, since \( C_1 \leq C_{1:2} \leq C_2 \), the ordering of the processors does not change. In the subsequent iteration, the equivalent processor and the processor with the next fastest communication link are selected and the steps are repeated until either all processors are used up, or Schedule \( g \) is found to be optimal. If Schedule \( g \) is found to be optimal in any iteration, then the algorithm exits after finding the load distribution for that iteration.

An example of how the sport algorithm works for a network with \( m = 3 \) is given in Fig. 6. In Fig. 6, Schedule \( l \) is found to be optimal for processors \( p_1 \) and \( p_2 \), and Schedule \( f \) is optimal for their equivalent processor \( p_{1:2} \) and the processor \( p_3 \). The resulting timing diagrams are as shown.

Now assume that this network has several more processors. Since sport adds processors one by one to the set of participants, Lemma 1 implies that the addition should be greedy, i.e. as many processors as possible should be used to minimize the processing time. If in the third iteration,
Schedule $g$ is found to be optimal for the processors $p_{1:3}$ and $p_k$, then in that case, the intervening time interval between the end of load allocation to $p_1$ and the start of result collection from $p_2$ would be zero. Since additional processors are always inserted (allocated load) within this interval, it would not be possible to allocate load to any more processors, and the algorithm exits. Thus Schedule $g$ forms the logical termination criterion for the algorithm unless all processors in the network are used. When Schedule $g$ is found to be the optimal schedule, all remaining processors are allocated zero load, and their index in the allocation and collection sequence is assigned zero value.

The computation of the allocation and collection sequences is straightforward. The allocation sequence $\sigma_a$ is maintained in the order of decreasing communication link bandwidth of the processors. Irrespective of the schedule found optimal in iteration $k$, $k$ is always appended to $\sigma_a$. The collection sequence $\sigma_c$ is constructed in the following manner:

- If Schedule $f$ or $g$ is found optimal in iteration $k$, $k$ is appended to $\sigma_c$.
- If Schedule $l$ is found optimal in iteration $k$, $k$ is prepended to $\sigma_c$.

The calculation of load distribution to the processors occurs simultaneously with the search for the optimal schedule. As shown in Fig. 7, the algorithm creates a binary tree of load fractions. If the number of processors participating in the computation is $n$, $2 \leq n \leq m$, the root node of the binary tree is $\alpha_{1:n}$, and the leaf nodes represent the final load fractions allocated to the processors. The value of the root node need not be calculated as it is equal to one. The individual load fractions, $\alpha_k$, are initially assigned value $\alpha'_k$ (say), and then updated at the end as:

$$\alpha_k = \begin{cases} \alpha'_k \cdot \prod^n_{j=2} \alpha_{1:j} & \text{if } k = 1 \\ \alpha'_k \cdot \prod^n_{j=k} \alpha_{1:j} & \text{if } k = 2, \ldots, n \end{cases}$$

This is equivalent to traversing the binary tree from the root to each leaf node and taking the product of the nodes encountered (see Fig. 7). This calculation can be easily implemented in $O(m)$ time by starting with the computation of $\alpha_{n}$, and storing the values of the product terms (i.e. $\prod \alpha_{1,j}$) for each processor and then using that value for the next processor.

Once the sequences ($\sigma_a, \sigma_c$) and load distribution $\alpha$ are found, calculating the processing time is trivial. The processing time is simply the sum of the (equivalent) parameters of the equivalent processor $p_{1:n}$, i.e., $T = C_{1:n} + E_{1:n} + \delta C_{1:n}$.

4.2 Complexity and Discussion

In sport, defining the allocation sequence by sorting the values of $C_k$ requires $O(m \log m)$ time, while finding the collection sequence and load distribution requires $O(m)$ time in the worst case. Thus, if sorted values of $C_k$ are given, then the overall complexity of the algorithm is polynomial in $m$ and is equal to $O(m)$.

The equivalent processor method works because:

- The equivalent processor maintains the positions of the processors in the allocation sequence in order of decreasing communication bandwidth (increasing value of communication parameter).
- Two processors in each iteration are kept immediately successive in the final ordering.
- All equations are linear. So irrespective of the final time interval $T$, the respective positions and load fractions are not changed.

It follows that if latencies are considered (i.e. computation and communication costs are affine functions of the size of data) in the system model, then the equivalent processor method may not work. However most applications of DLS where result collection is important, involve transfers of large volumes of data, and latencies can be safely ignored.

As noted earlier in Sect. 3.1, when result collection phase is considered along with heterogeneous networks, at this stage at least, it is not possible to a priori define any single sequence as the optimum sequence for allocation or collection. Nor is it possible to determine a criterion for the optimal number of processors to be used in the computation. The number of processors used and the optimal sequences depend on the communication and computation parameters of the processors and the application under consideration ($\delta$). Because of this, it is also not possible to derive closed-form equations for the load fractions or the total processing time.

SPORT does not guarantee a globally optimal solution to the DLSrchets problem. For example, if $C = \{10, 20, 30\}$, $E = \{5, 15, 25\}$, and $\delta = 0.5$, the globally optimal solution is $\sigma^*_a = \{1, 2, 3\}$, $\sigma^*_c = \{2, 3, 1\}$, and $\alpha^* = \{0.88, 0.08, 0.04\}$, while the solution found by sport is $\sigma_a = \{1, 2, 3\}$, $\sigma_c = \{2, 3, 1\}$. This shows that even though the algorithm guarantees a locally optimum solution, it does not necessarily guarantee a globally optimum solution.
The processors are first sorted in the order of decreasing $m$ below.

Finding the conditions for the minimization of the error is a part of the future work.

5. Simulation Results and Analysis

In the simulations, all $E_k$ and $C_k$ are defined in the same time units. On open networks such as the Internet, and Grid computing platforms, 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 [28]. 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 $\textit{sort}$, several simulations were performed with different ranges for $E_k$, $C_k$, and $\delta$.

5.1 Simulation Set A

The performance of $\textit{sort}$ was compared to four algorithms, viz. $\textit{opt}$, $\textit{fifo}$, $\textit{lifo}$, and $\textit{iterlp}$, each of which is described below.

The globally optimal schedule $\textit{opt}$ is obtained after evaluation of the linear program for all possible $(m!)^2$ permutations of $(\sigma_a, \sigma_c)$. The MATLAB™ linear program solver $\texttt{linprog}$ is used to determine the optimal solution to the linear program defined by constraints (1) to (4) 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 $\textit{opt}$ solution. This ensures that the minimal set of processors is used and the optimal processing time is found.

$\textit{lifo}$ and $\textit{fifo}$ heuristics are as follows. In $\textit{fifo}$, processors are allocated load and result are collected in the order of decreasing communication link bandwidth of the processors. In $\textit{lifo}$, load allocation is in the order of decreasing communication link bandwidth of the processors, while result collection is the reverse order of increasing communication link bandwidth of the processors.

For example let $C = \{20, 10, 15\}$, $E = \{5, 15, 10\}$. The processors are first sorted in the order of decreasing communication link bandwidth (i.e. by increasing value of $C_k$). The sorted processor numbers give the allocation sequence $\sigma_a = \{2, 3, 1\}$ for both $\textit{lifo}$ and $\textit{lifo}$. For $\textit{fifo}$, the result collection sequence, is the same as $\sigma_a$, i.e. $\sigma_c|_{\textit{fifo}} = \{2, 3, 1\}$. For $\textit{lifo}$, the result collection sequence is the reverse of $\sigma_a$, i.e. $\sigma_c|_{\textit{fifo}} = \{1, 3, 2\}$. For $\textit{fifo}$, the sequence pair $(\sigma_a, \sigma_c)$ so obtained, along with the sets $C$ and $E$, and $\delta$, are used to construct the linear program defined by constraints (1) to (4), which is passed to the linear program solver $\texttt{linprog}$, that determines the optimal $\textit{fifo}$ solution. For $\textit{lifo}$, using the transformation explained in [23], the optimal solution is found by using the closed form equations given in [14].

The $\textit{iterlp}$ heuristic 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 $(\sigma_a, \sigma_c)$ 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 $\sigma_a$ and $\sigma_c$. The next processor in the sequence is added in the next iteration. The new processor can be interleaved at any position in $(\sigma_a, \sigma_c)$, 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 $\Sigma_2^a = \{(3, 1, 2), (1, 3, 2), (1, 2, 3)\}$, and the set of possible collection sequences is $\Sigma_2^c = \{(3, 2, 1), (2, 3, 1), (2, 1, 3)\}$. 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. This is similar to $\textit{sort}$, except for the fact that equivalent processor cannot be used, and in the worst case, $\sum_{k=1}^m k^2 = O(m^3)$ linear programs have to be solved. It is computationally much too expensive to be used practically for large values of $m$, but can be used to compare the performance of other heuristic algorithms because of its near-optimal performance.

Preliminary simulations for other heuristic algorithms, viz. $\textit{fifo}$, $\textit{lifo}$, $\textit{fifo}$, $\textit{lifo}$, and $\textit{sumce}$, revealed large errors in favor of $\textit{sort}$, and it was decided not to pursue them further. The solutions to $\textit{fifo}$ and $\textit{lifo}$ are calculated similar to $\textit{fifo}$ and $\textit{lifo}$ except for the fact that the processors are not initially sorted. $\textit{sumce}$ and $\textit{sumce}$ distribute load fractions in the order of decreasing computation speed (i.e., increasing value of computation parameter, $E_k$). $\textit{sumce}$ 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 $C_k + E_k$).

To explore the effects of processor parameter values on the performance of the algorithms, several types of simulations were carried out as follows:

A1. $C$ homogeneous, $E$ homogeneous

Twenty-five different cases are considered, and the values for $C$ and $E$ in each case are obtained from the intervals $[C_{min}, C_{max}]$ and $[E_{min}, E_{max}]$ defined in Table 1. In each case, the intervals are partitioned into $m$ equal-sized, contiguous, non-overlapping sub-intervals of size $(C_{max} - C_{min})/m$ and $(E_{max} - E_{min})/m$. If these sub-intervals are represented as $I_{e1}, \ldots, I_{em}$ and $I_{c1}, \ldots, I_{cm}$, then for each case, $m^2$ sub-cases can be defined by taking the Cartesian product of the inter-
vals, i.e. \((I_{c1},I_{c2}), \ldots, (I_{cm},I_{cm})\). For each sub-case, the intervals are uniformly sampled to generate a homogeneous system, but also it is possible to compare a “fast” and (comparatively) “slow” homogeneous system.

A2. C homogeneous, E heterogeneous

Similar to Set A1, \(m\) sub-intervals \(I_{c1}, \ldots, I_{cm}\) are used for the communication parameters, but the computation parameters are generated by sampling the interval \(I_e = [E_{min}, E_{max}]\). This creates \(m\) sub-cases with intervals \((I_{c1},I_{c2}), \ldots, (I_{cm},I_{cm})\), such that in each sub-case, the communication parameters are homogeneous, but the computation parameters are heterogeneous.

A3. C heterogeneous, E homogeneous

This is the complement of Set A2, and the \(m\) sub-cases have intervals \((I_{c1},I_{c2}), \ldots, (I_{cm},I_{cm})\), where \(I_e = [C_{min}, C_{max}]\), such that the computation parameters are homogeneous and the communication parameters are heterogeneous.

A4. C heterogeneous, E heterogeneous

Similar to the previous sets, the sub-intervals \(I_{c1}, \ldots, I_{cm}\) and \(I_{c1}, \ldots, I_{cm}\) for each case are found. Each sub-interval is sampled once to generate a total of \(m\) values each for the communication and computation parameters. The values undergo a random permutation first before being assigned to the processors. Sampling the sub-intervals in this manner minimizes the possibility of two processors being allocated similar communication or computation parameters and generates a truly heterogeneous system. There are no sub-cases here.

For each set A1 to A4 mentioned above, simulations were carried out for \(m = 4, 5\) and \(\delta = 0.2, 0.5, 0.8\). For each variant algorithm, viz. opt, sport, iterlp, lifoc, and fifoc, at each value of \(m\) and \(\delta\), 100 simulation runs were carried out for each of the sub-cases in sets A1 to A3, and the 25 cases in Set A4. Let \(v = 1, \ldots, 4\) represent the four variant algorithms sport, iterlp, lifoc, and fifoc. The total processing time for each variant, \(T_v\), was calculated in each run and the percentage deviation from the optimal processing time, \(\Delta T_v\), for each variant was calculated as:

\[
\Delta T_v = \frac{T_v - T_{opt}}{T_{opt}} \times 100\% \quad v = 1, \ldots, 4
\]

Further processing and analysis is explained individually for each set below. Plots are to log-scale to magnify the values close to zero. Not all plots are shown on account of space considerations.

A1. C homogeneous, E homogeneous

In this set, for each case in Table 1, there are 16 and 25 sub-cases for \(m = 4\) and 5 respectively. The first sub-case represents a system with fast network links and high computation speed, while the last sub-case represents a system with slow network links and low computation speed. The intermediate sub-cases represent various other combinations of network and computation speed.

The cases in Table 1 are defined with intervals differing in both interval width (ratio) as well as the absolute values of the communication and computation parameters. To aggregate the performance obtained over all the intervals, the error values of the individual sub-cases should be averaged over the 25 cases in Table 1 to give the final mean percent error for each sub-case. Let \(i = 1, \ldots, 25\) represent the cases in Table 1, and \(j = 1, \ldots, m^2\) represent the sub-cases in each case. Then for case \(i\), sub-case \(j\), 100 simulation runs generate 100 values \(\Delta T_{ejk}\), \(k = 1, \ldots, 100\). These are averaged to get the mean error with respect to optimal, \(\overline{\Delta T_{ej}}\), for each variant:

\[
\overline{\Delta T_{ej}} = \frac{\sum_{k=1}^{100} \Delta T_{ejk}}{100} \quad i = 1, \ldots, 25, \quad j = 1, \ldots, m^2, v = 1, \ldots, 4
\]

Finally, for each variant, the values \(\overline{\Delta T_{ej}}\) are averaged over the 25 cases in Table 1, to compute the final mean percent error, \(\overline{\Delta T_{ej}}\) for that variant in each sub-case:

\[
\overline{\Delta T_{ej}} = \frac{\sum_{i=1}^{25} \overline{\Delta T_{ej}}}{25} \quad j = 1, \ldots, m^2, v = 1, \ldots, 4
\]

These values of \(\overline{\Delta T_{ej}}\) are plotted in Figs. 8 and 9 for \((m, \delta)\) pairs (4, 0.2) and (5, 0.8) respectively. Because the network links are homogeneous, fifoc is expected to perform well. It is observed that sport performs almost exactly the same as fifoc, with error between 0.1% and 0.01% \(\approx 0\). iterlp performance is even better, while lifoc has comparatively large error, and the error increases with increase in value of \(\delta\).

A2. C homogeneous, E heterogeneous

In this set, for each case in Table 1, there are 4 and 5 sub-cases for \(m = 4\) and 5 respectively. The first sub-case represents a system with fast network links, while the last sub-case represents a system with slow network links.
links. The computation speed is heterogeneous. The values of $\langle \Delta T_{ij} \rangle$ for the sub-cases are calculated as explained in set A1 by averaging over the 25 cases in Table 1. The only difference is that the value of $j$ ranges from 1, ..., $m$ instead of from 1, ..., $m^2$ as in set A1. Figs. 10 and 11 show the plots for $(m, \delta)$ pairs (4, 0.5) and (5, 0.2) respectively.

Again it is observed that as long as the network links are homogeneous, SPRT, ITERLP, and FIFOC are insensitive to the heterogeneity in the computation speed of the processors with average error between 0.1% to 0.001% $\approx 0$. On the other hand, errors in FIFOC persist and increase with $\delta$.

A3. C heterogeneous, $E$ homogeneous

As in set A2, this set too has 4 and 5 sub-cases for $m = 4$ and 5 respectively. The first sub-case represents a system with fast computation speed, while the last sub-case represents a system with slow computation speed. The network links are heterogeneous. The values of $\langle \Delta T_{ij} \rangle$ are calculated analogous to set A2. The plots for $(m, \delta)$ pairs (4, 0.8) and (5, 0.5) are shown in Figs. 12 and 13 respectively.

This simulation set clearly shows the adaptiveness of SPRT. FIFOC, which had almost zero error in the previous two sets, now has large error as compared to the optimal schedule. SPRT however, continues to have low error values around 0.1% along with ITERLP and ITERLP.

A4. C heterogeneous, $E$ heterogeneous

Since there are no sub-cases in this set, for each case $i = 1, \ldots, 25$, in Table 1, 100 simulation runs generate 100 values $\Delta T_{ik}$, $k = 1, \ldots, 100$. These 100 values are
Fig. 13 $\langle \Delta T \rangle$ in set A3 for $m = 5, \delta = 0.5$.

Fig. 14 $\langle \Delta T \rangle$ in set A4 for $m = 4$.

Fig. 15 $\langle \Delta T \rangle$ in set A4 for $m = 5$.

The final error values for each variant $\langle \Delta T_v \rangle$ are calculated by averaging over the 25 cases in Table 1.

$$\langle \Delta T_v \rangle = \frac{\sum_{i=1}^{25} \Delta T_{i,v}}{25} \quad v = 1, \ldots, 4$$

The values of $\langle \Delta T_v \rangle$ for $\delta = 0.2, 0.5, 0.8$ at $m = 4$ and 5 are shown in Figs. 14 and 15 respectively.

Sport, Lifoc, and Iterlp are seen adapt to the heterogeneity in the processor computation and network link speeds. The percent error of Fifoc increases with the increase in $\delta$, but there is a reduction in the error of the other three variants.

Though not strictly applicable, some trends can be identified:

- In set A1, for the same value of $\delta$, errors for “fast” homogeneous systems are higher than “slow” homogeneous systems.
- In sets A2 and A3, for the same value of $\delta$, errors for the “fast” and heterogeneous systems are higher than the “slow” and heterogeneous systems.
- In sets A1, A2, and A3, where either one or both of the variables are homogeneous, the average error increases with increase in $\delta$. However, in set A4, error reduces with $\delta$, for the better performing algorithms.

The minimum and maximum mean error values of each algorithm are tabulated in Tables 2 and 3. Scientific notation is used to enable a quick comparison of the algorithms in terms of orders of magnitude. It can be observed that overall in sets A1 and A2, the minimum and maximum errors in Fifoc are 2 orders of magnitude higher than Sport, Iterlp, and Lifoc. On the other hand in sets A3 and A4, Fifoc error is 2 to 3 orders of magnitude higher than the other three algorithms.

The extensive simulations carried out in Set A clearly show that:

- If network links are homogeneous, then Lifoc performance suffers for both homogeneous and heterogeneous computation speeds.
- If network links are heterogeneous, then Fifoc performance suffers for both homogeneous and heterogeneous computation speeds.
- Sport performance is also affected to a certain degree by the heterogeneity in network links and computation speeds, but since sport does not use a single predefined sequence of allocation and collection, it is able to better adapt to the changing system conditions.
- Iterlp performance is somewhat better than sport, but is expensive to compute. Sport generates similar schedules at a fraction of the cost.

5.2 Simulation Set B

To evaluate the performance of the algorithms with the increase in number of nodes, the processing times of Fifoc and Lifoc were compared with Sport. This is because, opt and Iterlp cannot be practically carried out beyond $m = 5$ and
Table 2 Minimum statistics for simulation set A.

<table>
<thead>
<tr>
<th>Set</th>
<th>m</th>
<th>δ = 0.2</th>
<th>δ = 0.5</th>
<th>δ = 0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>4</td>
<td>5.76e-03</td>
<td>8.08e-01</td>
<td>1.07e+00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.89e-04</td>
<td>6.90e-04</td>
<td>9.21e-01</td>
</tr>
<tr>
<td>A2</td>
<td>4</td>
<td>1.01e-02</td>
<td>1.01e-02</td>
<td>1.07e+00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.34e-03</td>
<td>7.93e-03</td>
<td>1.06e+00</td>
</tr>
<tr>
<td>A3</td>
<td>4</td>
<td>2.03e-03</td>
<td>1.05e+00</td>
<td>1.12e+00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.96e-01</td>
<td>1.90e-01</td>
<td>1.75e-01</td>
</tr>
<tr>
<td>A4</td>
<td>4</td>
<td>4.95e-06</td>
<td>4.92e-06</td>
<td>4.92e-06</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.08e-02</td>
<td>1.43e-02</td>
<td>1.15e-02</td>
</tr>
</tbody>
</table>

Table 3 Maximum statistics for simulation set A.

<table>
<thead>
<tr>
<th>Set</th>
<th>m</th>
<th>δ = 0.2</th>
<th>δ = 0.5</th>
<th>δ = 0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>4</td>
<td>5.34e-02</td>
<td>5.13e-02</td>
<td>3.39e-01</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8.79e-02</td>
<td>1.12e-01</td>
<td>1.07e-01</td>
</tr>
<tr>
<td>A2</td>
<td>4</td>
<td>5.36e-02</td>
<td>9.35e-02</td>
<td>5.13e-02</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.27e-01</td>
<td>1.27e-01</td>
<td>3.68e-01</td>
</tr>
<tr>
<td>A3</td>
<td>4</td>
<td>4.01e-01</td>
<td>4.03e-01</td>
<td>4.03e-01</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.26e-01</td>
<td>1.27e-01</td>
<td>1.26e-01</td>
</tr>
<tr>
<td>A4</td>
<td>4</td>
<td>1.32e+00</td>
<td>8.02e-01</td>
<td>1.12e+00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.56e+00</td>
<td>9.35e-01</td>
<td>1.15e+00</td>
</tr>
</tbody>
</table>

$m = 10$ respectively. Using the procedure used in simulation Set A4, 100 simulation runs were carried out for sport, lifoc, and fifoc, at $m = 10, 50, 100, \ldots, 300, 350$, and $\delta = 0.2, 0.5, 0.8$ for each of the 25 cases listed in Table 1. $\Delta T_v$, for each variant $v$ (lifoc $:= 1$ and fifoc $:= 2$) was found as:

$$\Delta T_v = \frac{T_v - T_{\text{SPORT}}}{T_{\text{SPORT}}} \times 100\%$$

$\nu = 1, 2$

Mean error, $\overline{\Delta T}_v$, for each case $i = 1, \ldots, 25$ in Table 1 was calculated by averaging $\Delta T_{v,ik}, k = 1, \ldots, 100$, over the 100 simulation runs and plotted.

Figure 16 shows the plots for $\overline{\Delta T}_v$ at $m = 10$, $\delta = 0.2, 0.5, 0.8$. First of all, lifoc is seen to always have a positive error with respect to sport. This is to be expected since the system is heterogeneous. The value of error increases with increase in the value of $\delta$.

Secondly, lifoc has a negative error with respect to sport for several cases at $\delta = 0.2$, i.e., the processing time of lifoc is smaller than sport. This is also to be expected since lifoc uses all available processors and every processor added reduces the processing time by some amount. This ends up distributing very tiny load fractions (smaller than $1 \times 10^{-6}$) to a large number of tail-end processors. As the value of $\delta$ increases, the error between lifoc and sport becomes insignificant.

This pattern of results is repeated even for higher values of $m$ as can be seen in Figs. 17 and 18 for $m = 100$ and 300 respectively. It can be observed that as the number of processors increases, fifoc performance in case numbers 11–15 and 21–25 becomes almost equal to that of sport. These ranges correspond to the intervals $I_i = [10, 100]$ and $I_c = [100, 1000]$ respectively, i.e., a ratio of $C_{\text{min}} : C_{\text{max}} = 1 : 10$. Because of the methodology used to perform the simulations, with a large number of processors, the values of $C_k$ tend to become similar to each other. Consequently, $(C_T - C_{\text{fifoc}})$ in (A–18) becomes small, and Schedule $\sigma$ always tends to be optimal for each pair of processors being compared. If Schedule $\sigma$ is optimal for all processors in sport, the resulting $\sigma_{\text{fifoc}}$ and $\sigma_{\text{lifoc}}$ are the same as fifoc and lifoc. However, surprisingly, cases 1–5, with $I_c = [1, 10]$ do not show this trend. This leads us to hypothesize, that the performance of the algorithms not only depends on the range (ratio) of parameters but also on the absolute values of the parameters. This belief is reinforced by the fact that in case numbers 21–25, lifoc also has comparable performance to sport, especially at higher values of $\delta$ and $m$.

Consider Table 4 that gives the minimum error of lifoc with respect to sport, the case number when it occurs, along with the mean error of lifoc averaged over all 25 cases (i.e., $\overline{\Delta T}_v$ in set A4) for different values of $\delta$ and $m$. The minimum error for lifoc is $-5.76\%$ for $m = 100$, $\delta = 0.8$, case number 2, but the minimum average error is $-2.12\%$ for $m = 300$, $\delta = 0.5$. It can be observed that the average error values at $\delta = 0.5$ are all smaller than those at $\delta = 0.2$, while the average error values at $\delta = 0.8$ are again greater than those at $\delta = 0.5$ (except for $m = 250$). We hypothesize that initially as $\delta$ increases, the error increases, but as $\delta \to 1$, i.e., size of result data approaches the size of allocated load, performance of sport and lifoc becomes similar. This is supported by the results of set A4, where the $\overline{\Delta T}_v$ of sport and lifoc is almost equal for $\delta = 0.8$ (see Figs. 14 and 15).

There is a significant downside to lifoc because of its property to use all available processors — the time required to compute the optimal solution (CPU time) is more than an order of magnitude greater than that of sport as seen in Fig. 19. These values were obtained separately from the
simulations above by averaging the CPU time over 100 runs for $L_c = [10, 100]$, $L_e = [50, 500]$, and $\delta = 0.5$. The results show that though both $\text{sport}$ and $\text{lifoc}$ are $O(m)$ algorithms, clearly $\text{sport}$ is the better performing algorithm, with the best cost/performance ratio for large values of $m$. The CPU time values for $\text{fifoc}$ are more than three orders of magnitude larger than $\text{sport}$ — too large to even warrant consideration.

The other disadvantage of $\text{lifoc}$ is that the chain of multiplications involved in the calculation of load fractions quickly leads to underflow because the numbers involved are tiny fractions and multiplying them results in smaller and smaller numbers until the floating point system cannot handle them anymore. Because of this, for $m > 150$, it is difficult to get valid results for $\text{lifoc}$ in a large number of cases. For example, for $m = 250, 300, 350$, $\text{lifoc}$ returned underflow errors in 24, 32 and 32 runs respectively out of the 100 simulation runs carried out. In MATLAB™, this causes a NaN (Not A Number) to be returned, and the load fractions cannot be calculated. Of course this is not a limitation of the algorithm itself, nevertheless it is an important practical consideration during implementation.
6. Conclusion

In this paper, a polynomial time algorithm, **SPORT**, for the scheduling of divisible loads on heterogeneous processing platforms and considering the result collection phase is presented. A large number of simulations were performed and it is found that **SPORT** consistently delivers good performance irrespective of the degree of heterogeneity of the system, the number of nodes, or the size of result data. **SPORT** primarily uses **LIFO** and **FIFO** as its base, but instead of having a single predefined sequence, it iteratively builds a locally optimal solution leading to a low error value.

This paper includes, for the first time ever, (a) the derivation of the condition to identify the presence of idle time in a **FIFO** schedule for two processors, (b) the identification of the limiting condition for the optimality of **FIFO** and **LIFO** schedules for two processors, and (c) the introduction of the concept of **equivalent processor** in **DLSRCHETS**.

As future work, the conditions (constraints on values of $E_2$ and $C_1$), that minimize the error need to be found. An interesting area would be the investigation of the effect of affine cost models, processor deadlines and release times. Another important area would be to extend the results to multi-installment delivery and multi-level processor trees.

Acknowledgments

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

References


**Appendix: Derivations**

The equations for load fractions, processing times, and the conditions for optimality of Schedules $f$, $l$, and $g$ are derived in brief in A.1, A.2, and A.3. Several intermediate steps in the derivations are not shown on account of space considerations.

**A.1 Schedule $f$**

From Fig. 4(a), $\alpha_1^f(E_1 + \delta C_1) = \alpha_2^f(C_2 + E_2)$. Using $\alpha_1^f + \alpha_2^f = 1$ gives

$$\alpha_1^f = \frac{C_2 r_2^f}{C_1 r_1^f + C_2 r_2^f}$$  \hspace{1cm} (A·1)

$$\alpha_2^f = \frac{C_1 r_1^f}{C_1 r_1^f + C_2 r_2^f}$$  \hspace{1cm} (A·2)

where

$$r_1^f = \delta + \rho_1$$

$$r_2^f = 1 + \rho_2$$

It is interesting to see $\alpha_1^f$ and $\alpha_2^f$ as weighted ratios of $C_2$ and $C_1$. The weights are functions of the network as well as the application under consideration. The processing time of Schedule $f$ is

$$T^f = \alpha_1^f C_1 (1 + \delta + \rho_1) + \delta \alpha_2^f C_2$$

Using (A·1) and (A·2),

$$= \frac{C_1 C_2}{C_1 r_1^f + C_2 r_2^f} \left( (1 + r_1^f)(\delta + r_2^f) - \delta \right)$$  \hspace{1cm} (A·3)

It is advantageous to distribute load to the two processors in Schedule $f$ instead of processing it entirely on $p_1$, if $T^f \leq T^l$. Now, from (A·3),

$$T^f \leq T^l \iff \frac{C_1 C_2}{C_1 r_1^f + C_2 r_2^f} \left( \delta r_1^f + r_2^f + r_1^f r_2^f \right) \leq C_1 (1 + r_1^f)$$

$$\iff \delta C_2 \leq C_1 (1 + \delta + \rho_1)$$  \hspace{1cm} (A·4)

Similarly, it is advantageous to distribute load to the two processors in Schedule $f$ instead of processing it entirely on $p_2$, if $T^f \leq T^g$. Again using (A·3),

$$T^f \leq T^g \iff \frac{C_1 C_2}{C_1 r_1^f + C_2 r_2^f} \left( \delta r_1^f + r_2^f + r_1^f r_2^f \right) \leq C_2 (\delta + r_2^f)$$

$$\iff \delta C_1 \leq C_2 (1 + \delta + \rho_2)$$  \hspace{1cm} (A·5)

Equation (A·5) is always true for $C_1 \leq C_2$.

To derive the equation for equivalent processor of Schedule $f$, algebraic manipulation of (A·3) gives

$$T'_{1/2} = \frac{C_1 C_2}{C_1 r_1^f + C_2 r_2^f} \left( r_1^f + r_2^f + \delta r_1^f + \delta r_2^f \right.$$  \hspace{1cm} (A·6)

$$+ r_1^f r_2^f - r_1^f - \delta r_2^f + \delta - \delta \right)$$

$$= \frac{C_1 C_2 (r_1^f + r_2^f) + C_1 C_2 (\rho_1 \rho_2 - \delta)}{C_1 r_1^f + C_2 r_2^f}$$

$$+ \frac{\delta C_1 C_2 (r_1^f + r_2^f)}{C_1 r_1^f + C_2 r_2^f}$$

**A.2 Schedule $l$**

From Fig. 4(b), $\alpha_1^l E_1 = \alpha_2^l (C_2 + E_2 + \delta C_2)$. Using $\alpha_1^l + \alpha_2^l = 1$ gives

$$\alpha_1^l = \frac{C_2 r_2^f}{C_1 r_1^l + C_2 r_2^f}$$  \hspace{1cm} (A·7)

$$\alpha_2^l = \frac{C_1 r_1^l}{C_1 r_1^l + C_2 r_2^f}$$  \hspace{1cm} (A·8)

where
\[ r'_1 = \rho_1 \]  
\[ r'_2 = 1 + \delta + \rho_2 \]  
\[ (A\cdot9) \]
\[ (A\cdot10) \]

We note that
\[ r'_1 = r'_1 - \delta \]  
\[ r'_2 = r'_2 + \delta \]  
\[ (A\cdot11) \]
\[ (A\cdot12) \]

\[ r'_1 + r'_2 = r'_1 + r'_2 = 1 + \delta + \rho_1 + \rho_2 \]  
\[ (A\cdot13) \]

The processing time of Schedule \( l \) is
\[ T^l = \alpha_1^l C_1 (1 + \delta + \rho_1) \]  
Using (A-7),
\[ = \frac{C_1 C_2 r'_1 (1 + \delta + r'_1)}{C_1 r'_1 + C_2 r'_2} \]  
\[ (A\cdot14) \]

Using (A-11) and (A-12)
\[ = \frac{C_1 C_2 (1 + r'_1) (\delta + r'_1)}{C_1 r'_1 + C_2 r'_2 + \delta (C_2 - C_1)} \]  
\[ (A\cdot15) \]

Schedule \( l \) is advantageous instead of distributing load entirely to \( p_1 \), if \( T^l \leq T^l \). From (A-14),
\[ T^l \leq T^l \iff \frac{C_1 C_2 r'_1 (1 + \delta + r'_1)}{C_1 r'_1 + C_2 r'_2} \leq C_1 (1 + \delta + r'_1) \]  
\[ (A\cdot16) \]
\[ \iff 0 \leq C_1 \rho_1 \]

Equation (A-16) is always true. Similarly, Schedule \( l \) is advantageous as compared to processing entire load on \( p_2 \), if \( T^2 \leq T^l \). Again using (A-14),
\[ T^l \leq T^2 \iff \frac{C_1 C_2 r'_2 (1 + \delta + r'_2)}{C_1 r'_1 + C_2 r'_2} \leq C_2 r'_2 \]  
\[ (A\cdot17) \]
\[ \iff C_1 (1 + \delta) \leq C_2 (1 + \delta + \rho_2) \]

Equation (A-17) is always true for \( C_1 \leq C_2 \).

To find the limiting condition for the optimality of Schedules \( f \) and \( l \), the equations for \( T^f \) and \( T^l \) are compared. From (A-3) and (A-15),
\[ T^f \leq T^l \iff \frac{(1 + r'_1) (\delta + r'_2) - \delta}{C_1 r'_1 + C_2 r'_2} \leq \frac{(1 + r'_1) (\delta + r'_2) - \delta}{C_1 r'_1 + C_2 r'_2 + \delta (C_2 - C_1)} \]
\[ \iff \frac{C_1 C_2}{C_1 r'_1 + C_2 r'_2} (1 + r'_1) (\delta + r'_2) - \delta \]
\[ \leq \frac{C_1 C_2}{C_1 r'_1 + C_2 r'_2} (1 + r'_1) (\delta + r'_2) - \delta \]
\[ \iff T^f \leq \frac{C_1 C_2}{C_1 r'_1 + C_2 r'_2} (1 + \delta + \rho_1) \]  
\[ (A\cdot18) \]

Conversely, it can be easily proved that
\[ T^l \geq \frac{C_1 C_2}{(C_2 - C_1)} \iff T^l \geq T^l \]

To derive the equation for equivalent processor for Schedule \( l \), algebraic manipulation of (A-15) gives
\[ T^l_{1:l} = \frac{C_1 C_2}{C_1 r'_1 + C_2 r'_2 + \delta (C_2 - C_1)} (r'_1 + r'_2) \]
\[ + \delta r'_1 + \delta r'_2 + r'_1 r'_2 - r'_1 - \delta r'_2 + \delta \]

Using (A-11), (A-12), and (A-13),
\[ = \frac{C_1 C_2 (r'_1 + r'_2)}{C_1 r'_1 + C_2 r'_2 + \delta (C_2 - C_1)} \]
\[ + \delta C_1 C_2 (r'_1 + r'_2) \]
\[ C_1 r'_1 + C_2 r'_2 \]
\[ (A\cdot19) \]

\section*{A.3 Schedule \( g \)}

From Fig. 4(c), \( \alpha_1^g E_1 = \alpha_2^g C_2 \). Using \( \alpha_1^g + \alpha_2^g = 1 \) gives
\[ \alpha_1^g = \frac{C_2}{E_1 + C_2} \quad \alpha_2^g = \frac{C_2}{C_2 + C_1 \rho_1 + C_2} \]  
\[ (A\cdot20) \]
\[ (A\cdot21) \]

Idle time occurs in processor \( p_2 \) (i.e., \( \rho_2 \geq 0 \)), only when \( \alpha_2^g E_2 \leq \delta \alpha_1^g C_1 \). From (A-20) and (A-21),
\[ \alpha_2^g E_2 \leq \delta \alpha_1^g C_1 \iff \rho_1 \rho_2 \leq \delta \]  
\[ (A\cdot22) \]

The processing time of Schedule \( g \) is
\[ T^g = \alpha_1^g C_1 (1 + \delta + \rho_1) + \delta \alpha_2^g C_2 \]

Using (A-20) and (A-21),
\[ = \frac{C_1 C_2}{C_1 \rho_1 + C_2} (1 + \delta + \rho_1) \]  
\[ (A\cdot23) \]

It is advantageous to distribute load to the two processors in Schedule \( g \) instead of processing it entirely on \( p_1 \), if \( T^g \leq T^l \). From (A-23),
\[ T^g \leq T^l \iff \frac{C_1 C_2}{C_1 \rho_1 + C_2} (1 + \delta + \rho_1) \leq C_1 (1 + \delta + \rho_1) \]
\[ \iff \delta C_2 \leq C_1 (1 + \delta + \rho_1) \]  
\[ (A\cdot24) \]

Similarly, it is advantageous to distribute load to the two processors in Schedule \( g \) instead of processing it entirely on \( p_2 \), if \( T^g \leq T^2 \). Again using (A-23),
\[ T^g \leq T^2 \iff \frac{C_1 C_2}{C_1 \rho_1 + C_2} (1 + \delta + \rho_1) \leq C_2 (1 + \delta + \rho_2) \]
\[ \iff C_1 (1 + \delta + \rho_1) \leq C_2 (1 + \delta + \rho_2) \]  
\[ (A\cdot25) \]
Equation (A·25) is always true for $C_1 \leq C_2$.

To find the limiting condition for optimality of Schedules $g$ and $l$, the equations for $T^g$ and $T^l$ are compared. From (A·23) and (A·14),

$$T^g \leq T^l \iff \frac{(1 + \delta)(1 + \rho_1)}{C_1 \rho_1 + C_2} \leq \frac{(1 + \delta + \rho_1)^2}{C_1 r_1^l + C_2 r_2^l}$$

Using (A·9),

$$\iff C_2 r_2^l \rho_1 \leq C_1 \rho_1 (\delta r_2^l + \rho_2 + \rho_1 \rho_2)$$

Using (A·10),

$$\iff C_2 \leq C_1 \left(1 + \frac{(1 + \rho_1) \rho_2}{\delta (1 + \delta + \rho_2)}\right) \quad (A·26)$$

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.

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.

Hidenori Nakazato received his B. Engineering degree in Electronics and Telecommunications from Waseda University in 1982, and his M.S. 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 Telecommunication 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.

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. He is a member of IEEE, IPSJ, ITE, IIEEJ.