UACEE International Journal of Advances in Computer Science and its Applications – IJCSIA
Volume 3 : Issue 2 [ISSN 2250 – 3765]
Publication Date : 05 June 2013

A Multi-round Algorithm for Minimum Processor in Real-time Divisible Load Scheduling

Pegah Razmara
Faculty of Computing Universiti Teknologi Malaysia
Johor, Malaysia
pegah.razmara@gmail.com
Suriayati Chuprat
Advanced Informatics School (UTM AIS) Universiti Teknologi Malaysia International Campus
Kuala Lumpur, Malaysia
suria@ic.utm.my

Mimi Liza Abdul Majid
Faculty of Computing Universiti Teknologi Malaysia
Johor, Malaysia
mimiliza3@gmail.com
Iza'in Nurfateha Ruzan
Faculty of Computing Universiti Teknologi Malaysia
Johor, Malaysia
aienruzan@gmail.co

Abstract— **Using of parallel and distributed system has become more common. Dividing data is one of the big challenges in this type of systems. Divisible Load Theory (DLT) is one of the proposed methods for scheduling data in parallel and distributed systems. Recent research has applied divisible load theory in real-time scheduling and has been introduced as an alternative for multiprocessor scheduling.There are two type of scheduling algorithm in real-time divisible load theory(RT-DLT) which is known as single-round and multi-round algorithm. Most studies in this field are about distributing data in single-round algorithm. Unfortunately, multi-round algorithms are difficult to analyze and have received only limited attention in real-time concept. In this paper, we will determine the minimum number of processors needed to complete the job by its deadline in multi-round algorithm. The two algorithms are compared on linear programming based formulation and result show that multi-round algorithm can provide a significant improvement on minimum number of processor needed in comparison with single-round algorithm.**

Keywords: Real-time Scheduling, Divisible Load Theory, Linear Programming, Single-round algorithm, Multi-round Algorithm

I. Introduction

Divisible Load Theory (DLT) studies a new model of distributed systems. It assumes that each partition of the computations is small, and there are no dependencies between the each part of computations. Therefore, the workload can be divided into different parts arbitrarily, and these parts can be executed independently in parallel. The sizes of the load parts should be adjusted to the speeds of computation and communication that causes task execution finishes in the shortest possible time[1].

In other words, DLT seeks optimal strategies to split divisible loads into chunks/ fractions and send them to the processing nodes with the goal of minimizing the overall processing resources and completion time.

These days, many researches were done in DLT theory with different concept in cluster-based research computing facilities[2-4] There are two methods for distributing of this kind of load fractions data among processors. First method is sending assigned data to each worker's in one-step (single round). Second, one is sending processor's assigned data in multi parts (multiple rounds).

Different researches were done in single-round DLT strategy [5, 6] by some assumptions. One round system with blocking and non-blocking mode communication [7], system with different processor available time (SDPAT) [8], non-dedicated systems [9], and others are some examples of the single-round investigated models.

Single round strategy is easy to implement but it gives rise to significant idle time for almost all processors due to a processor can start computing only after receiving the entire load fraction assigning to it. In other words, if the application is data intensive, the processors may face long idle times while waiting for data transmission[10]. Furthermore, this long idle time causes to increase execution time and number of processing nodes.

One way to reduce this idle time is to send the load fraction in more than one round. Therefore, that processor can begin its computation earlier in time. In other word, multi-round strategy reduces the general idle time of the processors at the end of the load distribution by subdividing the data fractions more and recurring distributing them [10, 11].This strategy is more difficult to implement than single one since the root processor has to perform a large number of operation to prepare the data for transmission but it cause considerable improvement in time performance. Also when we have limited in resource ,memory or buffer size and the data file to be processed is very large[11]. However, these works only consider a single task and for online scheduling of multiple tasks, it could be more challenging [5]. Moreover, deciding the proper number of processors is one significant challenge in multi-round strategy.

Recent research[12,13] obtained exact solutions to the problem of determining the proper number of processors that must be assigned to a job upon multiprocessor platforms, but it is not in real-time concept. In other words, they did not consider deadline in their solution. Therefore, in this paper we proposed another linear programming (LP) approach to determine the proper number of processors according to meet deadline in real-time multi-round algorithm.

This paper is organized as follows. In the next section, we present task model and system model. In Section III, a proposed method is presented which includes closed-form formula for finding the proper number of processors. In section IV, simulation result of our algorithm is discussed. Finally, section V gives the conclusions.

II. Task and system model

In this section, we describe task and system model in real-time divisible load theory, which is used in this research. In this method, we used client-server topology for network which all

processors are connected to a head processor and the head processor only schedule tasks and distribute chunks among workers and does not participate in computation.

In other words, the role of head node is accept or reject incoming jobs and execute the scheduling algorithm .This model includes of homogeneous environment, which means that all processing nodes have the same computational capacity and all links from head processor to workers have same bandwidth. This system does not have communication and computation over heads.

Moreover, it is assumed that data transmission does not happen in parallel. It means that head processor may be sending data to at most one worker at any time instant. However, in different processing nodes computation may accrue in parallel. Also, the head processors and workers are preemptive : the head processor completes the scheduling one job before considering the next job, and each workers complete computing one chunk of job before moving on to the other chunk of job that may have been assigned to it [14-16].

In RT-DLT each job is identified by a 3-tuple ( )

where >0 is job arrival time, >0 is total data size of the

job, and >0 is job relative deadline.

Table 1. Notations

Notation	Description
σ	Total size of data
V	Total size of each round
n	Number of processors
m	Number of rounds
A_i	Job arrival time
D_i	Job deadline
Δ	Time between current instant and deadline
C_m	Communication time
C_p	Computation time
α_i	Workload fraction
r_i	Ready time which is available time for each worker
s_i	Start time for receiving data from head node

III. Proposed method

In this research, we assume that all processors are available at same time. Under this model of processor availability, it is known that the completion time of a job on a given set of processing nodes is minimized if all the processing nodes complete their execution at the same instant. In other words, if some processing nodes complete the processing of a given workload before others then they will face idle time.
Moreover, in this research we assume that the size of all rounds is equal. If we change the size of rounds and increase that, the idle time between each round is increased. If we decrease the round size, data-transferring time to the previous processor is decreased, but current processing node is busy since computation time of previous round is large. For a given job ( ) and given number of processor, denote the amount of the workload that is assigned to the i^{th} processing node *1 i n*.

The primary idea in this algorithm is according to the first principles. In this algorithm we started out with no processors and continually added to them until processing the job finished (line 8 in the pseudo-code), or we specify that it is not possible to schedule this job by its deadline (line 9).
In greater detail, we are given the total size of the workload ( ), the amount of time between the current instant and the deadline ( ), the computation time and communication time which are cluster parameters, and the ready times for each processor , ,…, in regular order. The minimum number of processors needed ( ) have been determined, the fractions allocated to each processor (the 's), and the start time at that each processor will begin receiving data from the head node (the 's) .

```
MINPROCS(σ,Δ)
1. read numround;
2. minproc ←0 ;
3. σ ← ( σ ÷ numround);
4. Δ ← ( Δ ÷ numround);
5. for round=1 to numround do
6.    $s_1 \leftarrow r_1$ ; sum←0; i←1 ; D ← Δ;
7.    while (true) do
8.       if ( sum ≥ 1) break end if
9.       if ( $s_i$ ≥ D ) break end if
10.      $\alpha_i \leftarrow (D - s_i) \div (\sigma \times (C_p + C_m))$
11.      $s_{i+1} \leftarrow \max(r_{i+1}, s_i + (\alpha_i \times \sigma \times C_m))$
```

```
12.     sum←sum+$\alpha_i$
13.     i←i+1
     end while
14.  if ( sum ≥ 1) then  success!!
15.      minproc ←max( minproc , i)
     else  cannot meet the deadline
17.       minproc ← ∞
     end if
18.    D ← D + Δ;
    end for
```

Fig. 1. Computing

The pseudo-code uses some additional variables: ***numround*** which is the number of rounds will be used in this algorithm ,***sum*** which is total portion size of the workload that has already been allocated to each workers, ***i*** that point to and the number of processors and ***D*** which is allocated deadline for each round. The main body of the pseudo-code is *for* loop which count the number of rounds and inside of that is an infinite while loop which the only exist reason is be one of two break statements. The break in line 8 shows that we have allocated the whole job to the exact number of processors and break execute in line 9 means that it is not possible to execute this job according to meet this deadline. For example, the number of processors with the different ready time are not enough to process the job according to meet its deadline.

If neither break statement executes, which is the faction of the job that is allocated to processor is calculated. The value is executed by time unit ( ) for receiving data fraction from the head node and computing this data for ( ) time unit. In optimal situation, we would like these processing nodes complete execution at the job deadline like time-instant *D*. Due to may only start receiving data at time-instant , we need to + + = *D* and we calculate value of in line 10.

Once computed the allocated fraction , we can calculate the time at which may start execution. This time is the later of ready time and the time at that has received data. Moreover the head-node is able to transmit the data fraction to . This computation of is done in line 11. Moreover, Lines 12 and 13 update the total values of the workload portion that already has been allocated, and the processor index, which is considered next. To determine the final number of processor in algorithm we should select maximum of them in each round, which is done in line 15. Furthermore, deadline increase in each round in line 18.

IV. Simulation results

In this section we have presented our simulation experiments and displayed some of our result and compare that with single-round algorithm in the same condition. Our experiment were performed in MATLAB and using linear-programming solving which is available with MATLAB to solve our LPs.

The outcomes of our experiments are plotted in Figure 2. For greater detail, we also present the results data in Table 2.

The graph in Figure 2 plot the minimum number of processors (n_{min}) required to complete a given real-time workload by its allocated deadline, when this minimum number of processors is computed by our multi-round algorithm (depicted in the graphs by red line) and when it is computed by the single-round algorithm [17] (depicted in the graphs by blue line). As can be seen in the graphs, typically the performance of our algorithm is better than single-round algorithm performance [17].

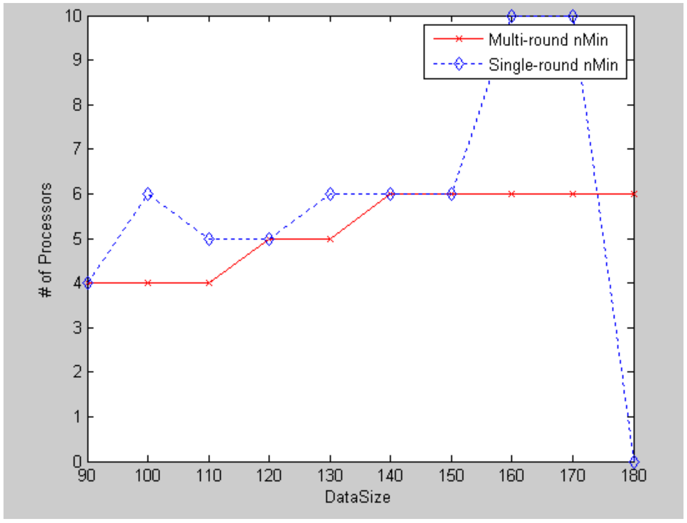
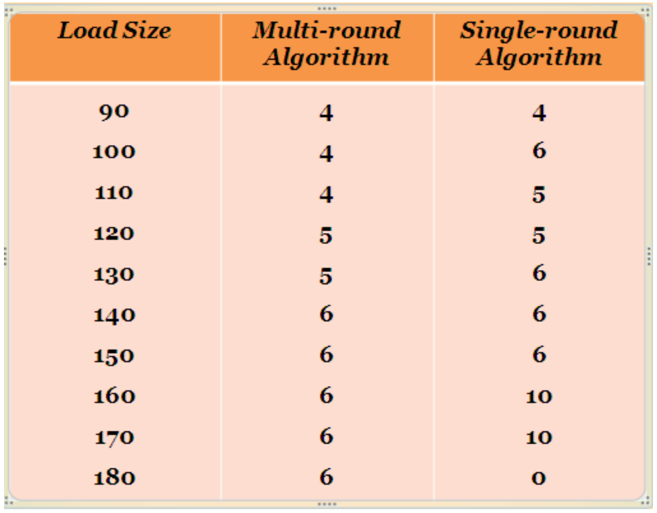


Fig. 2. Evaluation of produced $_i$ with increasing data size and cluster of n=16 processors.

The graphs in Figure 2 calculate the respective performance of the two algorithms since the size of the workload is increased, for 16 processors in cluster platform relatively. The performance enhancement is presented to be insignificant or very small for loads with small size; but when the load size increases, the performance penalty rewarded by the single-round algorithm[14, 17] becomes more considerable.

For better detail, we presented the minimum number of processors generated by both algorithms in Table 2.

Table 2. evaluation of produced with increasing datasize and a cluster of n=16 processors.



Load Size	*Multi-round Algorithm*	*Single-round Algorithm*
90	4	4
100	4	6
110	4	5
120	5	5
130	5	6
140	6	6
150	6	6
160	6	10
170	6	10
180	6	0

The advanced conclusion to be drawn from these experiments are that the previous single-round algorithm [17] is acceptable upon clusters in which workloads size are small and/or have large relative deadlines . In other words, for large size workloads according to our finding, single-round algorithm will not be able to execute the job and optimal multi-round algorithm performs significantly better.

V. Conclusion

In this paper, we have studied scheduling problems in RT-DLT. Moreover, some of fundamental characteristic in two models of divisible load scheduling, single-round and multi-round are presented. In addition, we proposed a multi-round algorithm that efficiently determines the minimum number of processors which are required to meet a job deadline and significantly decrease the number of processor in comparison with single-round algorithm in the same situation. In other word, through experimental evaluation, we have shown that this efficient algorithm significantly improves on the heuristic approximations proposed in single-round algorithm. As we mentioned above, designing multi-round algorithms in real-time concept is complex and less results are available in literature. In this paper we found a solution for one of them. There are two other significant challenges in multi round strategy which are proper number of rounds and scheduling the last round which in the future research we will be work on them.

Refrences

[1] M. Lawenda, "Multi-installment divisible loads scheduling," *university of technology*, 2006.

[2] A. Mamat, Y. Lu, J. Deogun, and S. Goddard, "Scheduling real-time divisible loads with advance reservations," *Real-Time Systems*, pp. 1-30, 2012.

[3] A. Mamat, Y. Lu, J. Deogun, and S. Goddard, "Efficient real-time divisible load scheduling," *Journal of Parallel and Distributed Computing*, 2012.

[4] A. Mamat, Y. Lu, J. Deogun, and S. Goddard, "An efficient algorithm for real-time divisible load scheduling," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, 2010, pp. 323-332.

[5] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, pp. 63-68, 2003.

[6] A. Shokripour and M. Othman, "Categorizing researches about DLT in Ten groups," in *Computer Science and Information Technology-Spring Conference, 2009. IACSITSC'09. International Association of*, 2009, pp. 45-49.

[7] O. Beaumont, A. Legrand, and Y. Robert, "Optimal algorithms for scheduling divisible workloads on heterogeneous systems," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003, p. 14 pp.

[8] A. Shokripour, M. Othman, and H. Ibrahim, "A new algorithm for divisible load scheduling with different processor available times," *Intelligent Information and Database Systems*, pp. 221-230, 2010.

[9] A. Shokripour, M. Othman, H. Ibrahim, and S. Subramaniam, "A new method for job scheduling in a non-dedicated heterogeneous system," *Procedia Computer Science*, vol. 3, pp. 271-275, 2011.

[10] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi, *Scheduling divisible loads in parallel and distributed systems* vol. 8: Wiley-IEEE Computer Society Press, 1996.

[11] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Cluster Computing*, vol. 6, pp. 7-17, 2003.

[12] A. Shokripour, M. Othman, H. Ibrahim, and S. Subramaniam, "A method for scheduling heterogeneous multi-installment systems," *Intelligent Information and Database Systems*, pp. 31-41, 2011.

[13] A. Shokripour, M. Othman, H. Ibrahim, and S. Subramaniam, "New method for scheduling heterogeneous multi-installment systems," *Future Generation Computer Systems*, 2012.

[14] X. Lin, Y. Lu, J. Deogun, and S. Goddard, "Real-time divisible load scheduling for cluster computing," in *Real Time and Embedded Technology and Applications Symposium, 2007. RTAS'07. 13th IEEE*, 2007, pp. 303-314.

[15] X. Lin, Y. Lu, J. Deogun, and S. Goddard, "Enhanced real-time divisible load scheduling with different processor available times," *High Performance Computing–HiPC 2007*, pp. 308-319, 2007.

[16] X. Lin, Y. Lu, J. Deogun, and S. Goddard, "Real-time divisible load scheduling with different processor available times," in *Parallel Processing, 2007. ICPP 2007. International Conference on*, 2007, pp. 20-20.

[17] S. Chuprat and S. Baruah, "Scheduling divisible real-time loads on clusters with varying processor start times," in *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA'08. 14th IEEE International Conference on*, 2008, pp. 15-24.