Scheduling Divisible Workloads from Multiple Sources in Linear Daisy Chain Networks

Xinxin Liu, Han Zhao, and Xiaolin Li
Scalable Software Systems Laboratory
Department of Computer Science
Oklahoma State University
Stillwater, OK 74078, USA
Email: {xinxin, haz, xiaolin}@cs.okstate.edu

***Abstract*—This paper considers scheduling divisible workloads from multiple sources in linear networks of processors. We propose a two phase scheduling strategy to minimize the overall processing time of these workloads by taking advantage of the processor equivalence technique. A case study with two sources of workloads is presented to illustrate the general approach for multiple sources of workloads. At the first phase, following the equivalent processor model to represent the processors in-between two load sources, we derive recursive equations to obtain near-optimal workload distribution for all processors and the minimum processing time of the overall workloads. At the second phase, we propose an efficient algorithm to obtain near-optimal load distribution for all processors represented by the equivalent processor. The simulation results and analysis for various scenarios are presented to show the behavior and efficiency of the proposed method.**

***Index Terms*—Linear Daisy Chain, Divisible Load Theory, Processor Equivalence**

I. Introduction

During the past decade, divisible load scheduling and distribution techniques have been widely studied forming the divisible load theory (DLT). The load is defined to be "divisible" for computation and communication if it can be partitioned arbitrarily among processors and links. DLT Offers a tractable and realistic approach to scheduling that allows integrated modeling of computation and communication in parallel and distributed computing systems [18]. DLT has been successfully applied to a wide spectrum of applications and systems, including parallel and distributed image processing [5], [10], sensor networks [15], computational grids [19] , and bioinformatics [20], just to name a few. Intensive research efforts have been made to study scheduling problems with various constrains on different interconnection topologies [11], [8], [13]. However, most DLT researches focused on the single source load distribution because the complexity of optimal scheduling increases exponentially with the increase of load sources. The concept of processor equivalence was first proposed [17]. This inspires us to adopt this model to tackle multi-source workload scheduling problems. In this paper, we present a heuristic to solve the two divisible loads scheduling in a linear daisy chain network of homogeneous processors. The proposed approach can be extended easily to solve the more general scheduling problem with multiple sources in heterogeneous networks. Our approach can be summarized as two phases. First, it collapses the processors from the left source to the right source into one processor to derive the optimal load distribution with other processors. Second, an effective algorithm is proposed to solve the local distribution within the collapsed processor. Our contributions in this work are two folds: **(1)** We propose to use the processor equivalence to study the multi-source divisible load scheduling problem; **(2)** We propose an efficient algorithm to assign near-optimal workloads among processors in the collapsed equivalent processor cluster. Further, the experimental evaluation is give to demonstrate the efficiency of the proposed strategy.

The rest of the paper is organized as follows. Section II presents related work. The problem description and definitions notations are in section III. The specific solution is illustrated in Section IV, first with five processors example then extend to n processors. In section V we practice the theory with numeric results. The last part of the paper is the conclusion and the prospective of future work.

II. Related Work

Single Source Load Distribution–With single workload source, previous research efforts have successively solved the optimal scheduling problem and obtained closed-form solutions on different network topologies, including trees [3], hypercubes [6] and meshes [8]. Most of the works focus on homogeneous platforms, works on heterogeneous platforms can be found in [11], [4], [1]. Furthermore, more practical problems are considered in some researches: scheduling on network can have communication startup costs [7], and with buffer constraints [21]. For very large workload, processing time can be further reduced using multiple installment load distribution [2]. Overlap of computation with communication can be improved using multiple rounds divisible load scheduling. A novel multi-round algorithm called UMR [22] is developed with consideration of latencies and heterogeneous platforms.

Multiple sources Load Distribution–Compared with researches on single source load distribution, the multiple sources load distribution problem has not received sufficient attention until recent years. Ko and Robertazzi combined Markovian queueing theory and divisible load scheduling theory on multi-source load distribution in 2002 [16]. One year later, multiple sources grid scheduling with capacity constraints was studied [9]. Multiple sources scheduling for small size models using linear programming and closed form solutions were proposed respectively in 2004 and 2005 [14], [12].

Our paper re-approaches the multiple sources load scheduling problem via the processor equivalence technique. The research is conducted on linear daisy chain topology, and provides a new vision to multiple sources load scheduling solution.

III. Problem description

The situation we are considering involves a linear daisy chain of n processors, as shown in Fig.5. We refer to the n processors as P_1, P_2,...,P_n. Two divisible workloads come into or originate from any two of these processors. The workloads are denoted as L_1 and L_2. All the processors are assumed to have front-end, which means communication and computation can be proceed simultaneously. Further, we can assume that the network environment is homogeneous, that is, all the processors have the same computation capacity, and the link speeds between any two processors are the same. Thus, the problem becomes how to partition and schedule the workloads amongst these processors to get the minimum finish time.

We describe the load scheduling process as follows: after the divisible loads L_1 and L_2 originated at any two processors, they are divided into n load fractions, and distributed among all processors. Upon receiving their respective load fractions, the processors start processing their own workload and rely all the other fractions to the next processor at the same time.

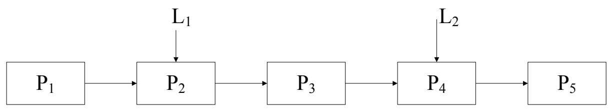

Fig. 1. Five processors Two load sources

To find a minimum time solution, all processors must stop computing at the exact same time instant; otherwise remaining workloads could be transferred from busy processors to the idle ones. The following notations and definitions are used:

Notations and Definitions

- P_i: the *ith* processor. i should be from 1 to N.
- P_{Δ_1}: the processor that originate work load L_1
- P_{Δ_2}: the processor that originate work load L_2
- P_{eq}: the equivalent professor of all the processors between processor$\Delta 1$ and $\Delta 2$, where the workload originated.
- L_i: the *ith* work load.Particually, i equals to 1 or 2 in our analysis, but it can be extended to x later on.
- a_i: the mount of workload that is assigned to the *ith*processor.
- ω_i: a constant that is inversely proportional to the processing speed of processor i in the chain.
- z_i: a constant that is inversely proportional to the speed of the *ith* link.
- T_i: finish time for the *ith* processor.
- T_{finish}: the finish time for the whole system, actually it will be equal to T_i.
- T_{cp}: normalized solution time when ω_i=1.
- T_{cm}: normalized transmission time when z_i=1.
- ΣL_i = 1

IV. Two Case Study

A. *Five Processors, Two Load Sources*

First we consider the five processors case. We assume, without lose of generality, that the two workloads are originated at the second and the fourth processor respectively,that is Δ_1 is 2 and Δ_2 is 4. Please refer to Fig. 1 to see the simple five processors network.

1) ***Global Stage****:* To solve this scheduling problem, we divide it into two phases: first we collapse the three processors P_2, P_3, P_4 into one "big" processor P_{eq}, which has more powerful computation ability compared with the other processors. The concept of collapsing two or more processors into an equivalent processor was first proposed by Dr. Robertazzi in [17], they discussed the change of the processing constant: ω_i which is inversely proportional to the processing speed of the processor and gave a solution for evaluating the new ω_{eq}. After collapsing, the timing diagram for the remaining three processors is shown in Fig. 2.

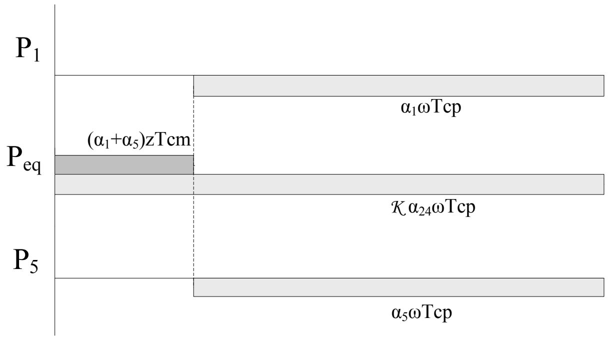

Fig. 2. Five processors timing diagram

Since mostly, the communication devices cannot guarantee that loads be transferred to both sides simultaneously, it is reasonable to first transfer the loads to the "outer" processors which are on both the left and right side of the equivalence processor so that the marginal processors can start computing loads distributed to them quickly.

Based on the timing diagram, we can set the equations for the finish time of each processor to get the optimal fragment size of loads:

$$T_1 = \alpha_1 * \omega T_{cp} + (\alpha_1 + \alpha_5) * zT_{cm} \quad (1)$$

$$T_{24} = \alpha_{24} * \kappa * \omega T_{cp} \quad (2)$$

$$T_5 = \alpha_5 * \omega T_{cp} + (\alpha_1 + \alpha_5) * zT_{cm} \quad (3)$$

$$\alpha_1 + \alpha_{24} + \alpha_5 = L_1 + L_2 = 1 \quad (4)$$

According to the time diagram, we can see

$$T_1 = T_{24} = T_5 = T_{finish} \quad (5)$$

Thus, from $T_1 = T_5$, we get $\alpha_1 = \alpha_5$. So α_{24} can be illustrated as

$$\alpha_{24} = \frac{\omega T_{cp} + zT_{cm}}{\kappa \omega T_{cp}} * \alpha_1 + \alpha_1 * zT_{cm} \quad (6)$$

Now we get the optimal workload fractions distributed to each processor are:

$$\alpha_1 = \frac{\kappa \omega T_{cp}}{(2\kappa + 1)\omega T_{cp} + 2zT_{cm}} \quad (7)$$

$$\alpha_5 = \frac{\kappa \omega T_{cp}}{(2\kappa + 1)\omega T_{cp} + 2zT_{cm}} \quad (8)$$

$$\alpha_{24} = \frac{2zT_{cm} + \omega T_{cp}}{(2\kappa + 1)\omega T_{cp} + 2zT_{cm}} \quad (9)$$

Thus, T_{finish} will be

$$T_{finish} = \frac{2zT_{cm} + \omega T_{cp}}{(2\kappa + 1)\omega T_{cp} + 2zT_{cm}} * \kappa \omega T_{cp} \quad (10)$$

*2) **Local Stage:*** Next we will evaluate the optimal load fractions assigned to those "middle" processors, which have been collapsed into one equivalent processor in the Global Stage. The remaining load can be transferred among these "middle" processors; but, it is hard to determine the direction of this kind of communication. As in Fig. 3, the workload can be passed from the leftmost processor to the rightmost one, or from right to the left, or even the two end will transfer both towards some processor in the middle. All the possible directions can be determined using the factor of workload amount at each end. Thus, we propose a local distribution method

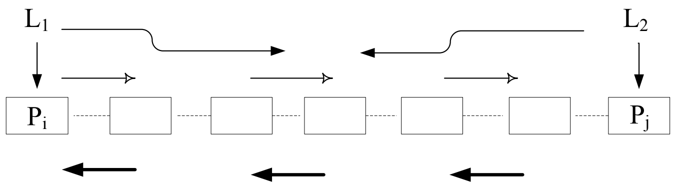

Fig. 3. Possible directions of communication

to transfer loads step by step. First by comparing the remained injected loads at left and right end processors, we can determine the direction of load transfer for the reason that it is meaningless to utilize the processor with less workload to provide load to those middle processors. Now we get the transferring timing diagram for processor P_2–the left end processor of the whole equivalent processor, with global time formulas we can now calculate the optimal load fraction distributed to processor P_2. Next step we can compare the remaining part of left injected loads with loads injected from right to get the optimal loads distributed to the next processor. The steps can be repeated to solve the problem of these processors which are collapsed into one "big" processor in *Global Stage*.

To illustrate our method, take this five processors network as an example. In global stage, α_1 and α_5 have been calculated. Now we continue our scheduling by assuming that L_1-α_1≥L_2-α_2, let σ_2 to be the potion of total workload that P_2 gets from the remaining workload, Fig. 4 is the timing diagram P_2 holds:

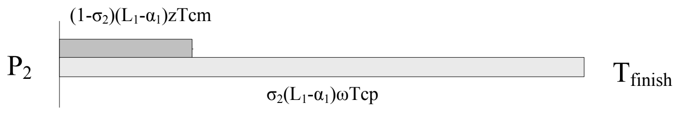

Fig. 4. Timing diagram for P_2

From

$$T_{finish} = \sigma_2(L_1 - \alpha_1)\omega T_{cp} \quad (11)$$

we get,

$$\alpha_2 = \sigma_2(L_1 - \alpha_1) = \frac{T_{finish}}{\omega T_{cp}} \quad (12)$$

The comparison steps are repeated until all the wanted load get processed and calculated.

B. N Processors, Two Load Sources

*1) **Global Stage:*** By collapsing from the processor originated left load source to the one with right load source, we could get one equivalent processor with a special computational ability: $\kappa\omega T_{cp}$. The timing diagram is shown in Fig. 7.

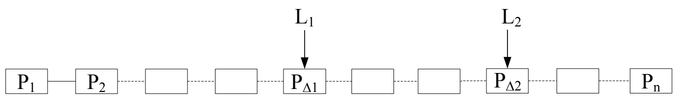

Fig. 5. N processors Two load sources

As T_i $(i = 1, 2, 3...n)$ represents the finish time of each processor, and T_{eq} is the finish time of the equivalent processor, we have these equations:

$$T_{eq} = \alpha_{eq}\kappa\omega T_{cp} \tag{13}$$

$$T_{eq-1} = \alpha_{eq-1}\omega T_{cp} + (\sum_{i=1}^{eq-1}\alpha_i + \sum_{i=eq+1}^{n}\alpha_i) * zT_{cm} \tag{14}$$

$$T_{eq+1} = \alpha_{eq+1}\omega T_{cp} + (\sum_{i=1}^{eq-1}\alpha_i + \sum_{i=eq+1}^{n}\alpha_i) * zT_{cm} \tag{15}$$

$$T_{eq-2} = \alpha_{eq-2}\omega T_{cp} + (\sum_{i=1}^{eq-1}\alpha_i + \sum_{i=eq+1}^{n}\alpha_i + \sum_{i=1}^{eq-2}\alpha_i) * zT_{cm} \tag{16}$$

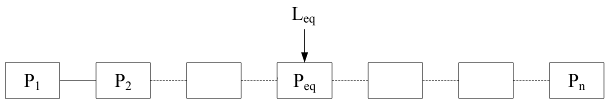

Fig. 6. N processors equivalent case

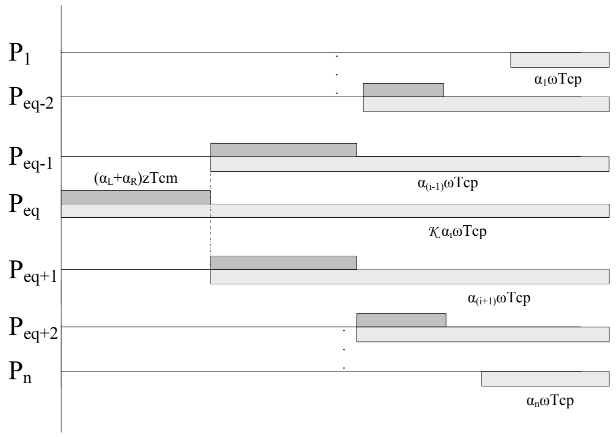

Fig. 7. N processors equivalent case

$$T_{eq+2} = \alpha_{eq+2}\omega T_{cp} + (\sum_{i=1}^{eq-1}\alpha_i + \sum_{i=eq+1}^{n}\alpha_i + \sum_{i=eq+2}^{n}\alpha_i) * zT_{cm} \tag{17}$$

$$T_{eq-3} = \alpha_{eq-3}\omega T_{cp} + (\sum_{i=1}^{eq-1}\alpha_i + \sum_{i=eq+1}^{n}\alpha_i + \sum_{i=1}^{eq-2}\alpha_i + \sum_{i=1}^{eq-3}\alpha_i) * zT_{cm} \tag{18}$$

$$T_{eq+3} = \alpha_{eq+3}\omega T_{cp} + (\sum_{i=1}^{i-1}\alpha_i + \sum_{i=eq+1}^{n}\alpha_i + \sum_{i=eq+2}^{n}\alpha_i + \sum_{i=eq+3}^{n}\alpha_i) * zT_{cm} \tag{19}$$

Continuing,

$$T_1 = \alpha_1\omega T_{cp} + (\sum_{i=1}^{eq-1}\alpha_i + \sum_{i=eq+1}^{n}\alpha_i + \sum_{i=1}^{1}\alpha_i) * zT_{cm} \tag{20}$$

$$T_n = \alpha_n\omega T_{cp} + (\sum_{i=1}^{eq-1}\alpha_i + \sum_{i=eq+1}^{n}\alpha_i + \sum_{i=n}^{n}\alpha_i) * zT_{cm} \tag{21}$$

Note the criteria that all processors must stop at the same time instant, the following condition will hold:

$$T_1 = T_2 = ... = T_n \tag{22}$$

With the condition of the sum of all the distributed loads equal to one, that is :

$$\sum_{i=1}^{n}\alpha_i = 1 \tag{23}$$

There are 2*(n-Δ_2+Δ_1) equations and 2*(n-Δ_2+Δ_1) unknowns, so these equations could be solved and get one deterministic solution set.

Let's begin from T_1 = T_2 we can get

$$\alpha_2 = \frac{\omega T_{cp} + zT_{cm}}{\omega T_{cp}} * \alpha_1 \tag{24}$$

And T_3 = T_2 yields

$$\alpha_3 = \frac{\omega T_{cp} + zT_{cm}}{\omega T_{cp}} * \alpha_2 + \alpha_1 zT_{cm} \tag{25}$$

TABLE I
ALGORITHM 1

```
***Initialization***
  $S_1$: Set includes $\alpha_1$ to $\alpha_{i-1}$
  $S_2$: Set includes $\alpha_{j+1}$ to $\alpha_n$
  $Index1$: Index number from 1 to $\Delta_1 - 1$
  $Index2$: Index number from $\Delta_2 + 1$ to $n$
  $S$ : Set includes all the optimal load from $\alpha_1$ to $\alpha_n$
  $Direction$: could be $Right$ or $Left$
  $temp$: temporary value for current index
***Local Distribution***
  Do {
 /* Comparing remaining load*/
   If (($L_1$-$\sum_{i=1}^{\Delta_1-1}\alpha_i$)$\geq$($L_2$-$\sum_{i=1}^{\Delta_2+1}\alpha_i$))
   {
       $Direction$ = $Left$;
       $temp$ = $MAX(i \in Index1)$+1;
 /*Find optimal load for current step*/
       Compute $\alpha_{temp}$;/* See Algorithm 2 */
       $\alpha_{temp} \longrightarrow S_1$;
    }
   Else {
 /*Compute from right side*/
        $Direction$ = $Right$;
       $temp$ = $MIN(i \in Index2)$-1
       Compute $\alpha_{temp}$;/* See Algorithm 2 */
       $\alpha_{temp} \longrightarrow S_2$;
   }
  } While($S_1 \bigcup S_2 \neq S$)
```

TABLE II
ALGORITHM 2

```
***Procedure Compute***
 ***Initializaion***
   $\sigma_i$: optimal fraction of remaining load distributed to
 $i - th$ processor
   $t$: index variable
 ***BEGIN***
  If ($Direction$ == $Left$)
   {
     $t$ = $temp - i$;
     If($t$ == 0){
       $T_{finish}$ =$\sum_{i=1}^{\Delta_1-1}\alpha_i$z$T_{cm}$+$\sigma_i$($L_1$- $\sum_{k=1}^{\Delta_1-1}\alpha_k$)$\omega T_{cp}$
     }
     Else /*t$\geq$1*/ {
       $T_{finish}$=$\sum_{k=1}^{\Delta_1-1}\alpha_k T_{cm}$+ [(1-$\sigma_i$)+(1-$\sigma_i$- $\sigma_{i+1}$)+
                 ...+(1-$\sigma_i$-$\sigma_{i+1}$-...-$\sigma_{i+t-1}$)] ($L_1$-$\sum_{k=1}^{\Delta_1-1}\alpha_k$)z$T_{cm}$
                 +$\sigma_{i+t}$($L_1$-$\sum_{k=1}^{\Delta_1-1}\alpha_k$)$\omega T_{cp}$;
       $\alpha_{temp}$ = $\alpha_{i+t}$($L_1$-$\sum_{k=1}^{\Delta_1-1}\alpha_k$);
         }
  Else { /*direction is right*/
     $t$ = $j - temp$
     If(t==0)
       $T_{finish}$=$\sum_{k=\Delta_2+1}^{n}\alpha_k$z$T_{cm}$+$\sigma_j$($L_2$-$\sum_{k=\Delta_2+1}^{n}\alpha_k$)$\omega T_{cp}$
     Else /*t$\geq$1*{
       $T_{finish}$ = $\sum_{k=\Delta_2+1}^{n}\alpha_k$z$T_{cm}$+ [(1-$\sigma_j$)+(1-$\sigma_j$- $\sigma_{j-1}$)+
                  ...+(1-$\sigma_j$-$\sigma_{j-1}$-...- $\sigma_{j-t+1}$)]($L_2$-$\sum_{k=\Delta_2+1}^{n}\alpha_k$)z$T_{cm}$
                  +$\sigma_{j-t}$($L_2$-$\sum_{k=\Delta_2+1}^{n}\alpha_k$)$\omega T_{cp}$
       $\sigma_{j-t}$($L_2$-$\sum_{k=\Delta_2+1}^{n}\alpha_k$) $\longrightarrow$ $\alpha_{temp}$
   } }
 ***END***
```

Continuing with this kind of deriving, we can get a uniform equation for these processors from P_1 to P_{eq}, that is $i \in$ (1,2,...,eq-1). For i = 1,2 we have

$$\alpha_1 = \alpha_1 \tag{26}$$

$$\alpha_2 = \frac{\omega T_{cp} + zT_{cm}}{\omega T_{cp}} * \alpha_1 \tag{27}$$

For i is greater than 2,

$$\alpha_i = \frac{\omega T_{cp} + zT_{cm}}{\omega T_{cp}} * \alpha_{i-1} + \sum_{j=1}^{i-2} \alpha_j zT_{cm} \tag{28}$$

As for P_{eq}, the workload is

$$\alpha_{eq} = \frac{\omega T_{cp} + zT_{cm}}{\kappa \omega T_{cp}} * \alpha_{eq-1} + \sum_{j=1}^{eq-2} \alpha_j zT_{cm} \tag{29}$$

Recursively substituting α_i with α_1, we can get these i fractions for P_1 through P_{eq} represented by α_1.

On the other side, if we start from P_{eq+1} to P_n, we also have

$$\alpha_i = \frac{\omega T_{cp} + zT_{cm}}{\omega T_{cp}} * \alpha_{i+1} + \sum_{j=i+2}^{n} \alpha_j zT_{cm} \tag{30}$$

Especially for P_{n-1} and P_n

$$\alpha_{n-1} = \frac{\omega T_{cp} + zT_{cm}}{\omega T_{cp}} * \alpha_n \tag{31}$$

$$\alpha_n = \alpha_n \tag{32}$$

Thus, for P_{eq}

$$\alpha_{eq} = \frac{\omega T_{cp} + zT_{cm}}{\kappa \omega T_{cp}} * \alpha_{eq+1} + \sum_{j=eq+2}^{n} \alpha_j zT_{cm} \tag{33}$$

Using substitution method, those $(n - eq)$ fractions from P_{eq} to P_n can be represented by α_n. But, as there is only one α_{eq}, two different representations will give a equation that allows using α_1 to represent α_n.

Back to the very beginning, as the total workload is 1, and all these fractions α_i $(i = 1, 2, 3...n)$ are represented by α_1, we can easily solve them.

2) ***Local Stage****:* As said before, the idea to solve the local case needs comparison at each step, and can be expressed in the following pseudo code algorithms. For clarity, a description of the variables used can be found in initialization step, see Table. I Algorithm1.

To compute the current optimal workload fractions, the near-optimal finish time calculated in global stage is used, and we get the following sub-procedure in Table. II Algorithm2.

The algorithm we proposed is basically iterative. At each step, the algorithm use those load fractions calculated in previous steps and compute the finish time of current processor, which consist of both the transmission time of previous workload and the processing time itself, to get the load fraction for the current one. This time should hold the equality of global minimum finish time, which is calculated in global stage. The algorithm terminates when all the middle processors get processed by the algorithm. The numerical example analysis is given out in section V.

C. *Extend to multiple sources problem*

The approach proposed in the general case can be easily extended to n processors, m sources problem. Suppose we have m workloads to be solved among n processors. We can collapse two processors at a time, and using the algorithm for local distribution analysis. In this way the problem of multiple sources on n processors can be solved.

V. Simulation Results

In this section, we first show that our load scheduling result can be influenced significantly by different factors. By highlighting all the important issues, we establish three sets of experimental data, and use the algorithm mentioned before to obtain the results. Second we compare our approach with a simple schema of load distribution–partition load equally amongst all the processors. The comparison result is satisfying for the reason that, with the number of processors increasing, our method effectively decreases the finish time of transferring and processing load.

A. *Factors to be Considered*

The problem of partitioning multiple load sources between n processors can be greatly influenced by various factors, such as the amount difference between load sources, ratio of computation ability to transmission ability, and the position where the workload originated in the system. All these factors make the problem much more complicated than a single workload scheduling. To illustrate this, we take three examples to show how different factors can influent the scheduling method. In the first example we employ a linear daisy chain consist of seven processors as standard. L_1 is 0.6 and Δ_1 is 3; L_2 is 0.4 and Δ_2 is 5. By setting $\omega_i T_{cp}$=4$z_i T_{cm}$, we can collapse the three processors in the middle into a equivalent one using the front-end processor solution from [17]. Since the model we used is a little different from theirs, the result is intuitively not accurate, but still near-optimal. After collapsing these middle processors, we can use our global and local strategy to obtain a solution. In the second case we change our processor number to 8, and change the ratio of computation ability to transmission ability to 10 while keeping all the other factors the same. At last we maintain the number of processors to be 8 and change the amount of workload to be $L_1 = L_2 = 0.5$. Also we change the position where two loads originated, let $\Delta_1 = 4$ and $\Delta_2 = 7$. The result is shown in Fig. 8

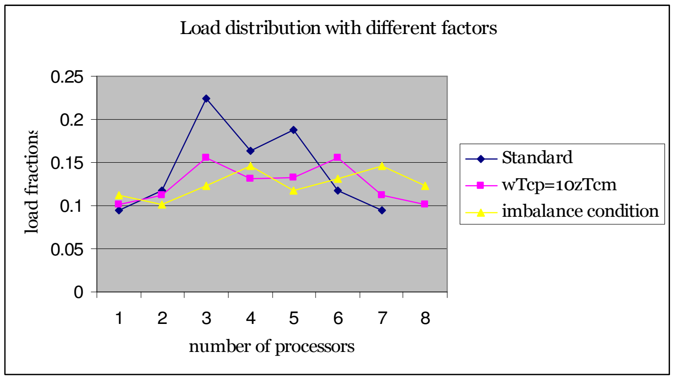

Fig. 8. Numerical Experiment 1

By carefully observing the figure we can come to a conclusion that when computational ability is much greater than the overhead of transferring, the processor which holds the original workload tends to not distribute the work to other processors thus making the load distribution more unbalanced. Another observation is that workload distribution relied on specific loads injected positions; different positions can result in different scheduling methods.

B. *Finish Time Improvement Using our Approach*

In this part, we compare our approach with a simple load distribution schema which partition the loads evenly amongst processors. Keeping the number of processors on each side of the two workloads balanced, we let $L_1 = 0.6$ and $L_2 = 0.4$. The ratio of computational ability to transferring ability is maintained invariant to 10. We calculated the specified results while the number of processors increasing. The result is shown in Fig. 9

From the figure we can find that, comparing with the equally partitioning method by which the finish time increases drastically while the number of processors increases, our approach efficiently decreases the finish time and generates a near-optimal solution. For example, when processor number comes to 10, our algorithm reduces time used in transferring and processing of the equal partition method by 55%. This is expected to improve more with the number of processors increasing.

VI. Conclusion and Future Work

The approach to solve the scheduling problem of multiple workloads in linear daisy chain is outlined in

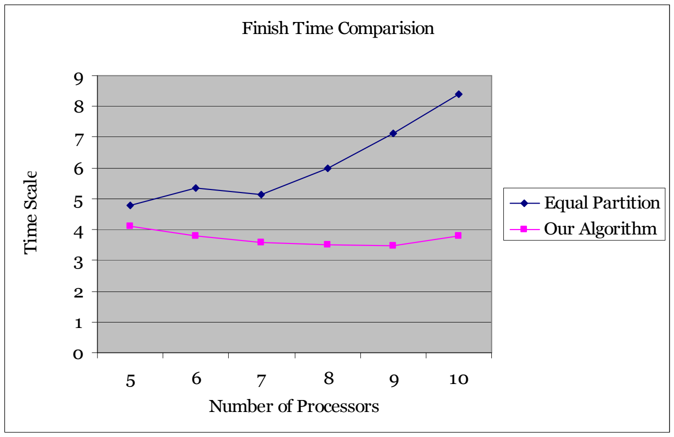

Fig. 9. Numerical Experiment 2

this paper. It is inspired by the concept of collapsing several processors into one and is proved to be effective to solve such problem. It is heuristic to apply the strategy to a specific problem in two steps: a recursive form solution can be used to compute outside the equivalent processor, and inside the equivalent processor an algorithm is designed to get optimal workload distribution for each processor. Various factors that could influent the load distribution are shown in the numerical experiment.

In the future, a more accurate solution for the equivalence of two workloads model is expected. Furthermore, effective application to solve the multiple loads distribution based on the algorithm provided in this article can be implemented.

References

[1] O. Beaumont, A. Legrand, and Y. Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 98.2, Washington, DC, USA, 2003. IEEE Computer Society.

[2] Ghose D. Bharadwaj, V. and V Mani. Multi-installment load distribution in tree networks with delays. *IEEE Transactions on Aerospace and Electronic Systems*, 31(2):555–567, 1995.

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

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

[5] Veeravalli Bharadwaj, Debasish Ghose, and Thomas G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, 2003.

[6] J. Blazewicz and M. Drozdowski. Scheduling divisible jobs on hypercubes. *Parallel Computing*, 21:1945–1956, 1995.

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

[8] M. Drozdowski and W. Glazek. Scheduling divisible loads in a three-dimensional mesh of processors. *Parallel Computing*, 25(4):381–404, 1999.

[9] Bharadwaj Veeravalli Han Min Wong, Dantong Yu and Thomas G. Robertazzi. Data intensive grid scheduling: Multiple sources with capacity constraints, Nov. 2003 2003.

[10] Bhuyan L. Zhiyong Xu Jingnan Yao, Jiani Guo. Scheduling real-time multimedia tasks in network processors. *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, 3, 2004.

[11] Jee G.-I. Kim, H.J. and Lee J.G. Optimal load distribution for tree network processors. *IEEE Transactions on Aerospace and Electronic Systems*, 32(2):607–612, 1996.

[12] T. Robertazzi M.Moges and D. Yu. Divisible load scheduling with multiple sources: Closed form solutions, 2005 2005.

[13] M. Mogas and Thomas G. Robertazzi. Optimal divisible load scheduling and markov chain models, 2003.

[14] M. Mogas and Thomas G. Robertazzi. Grid scheduling divisible loads from multiple sources via linear programming, 2004.

[15] Robertazzi T.G. Moges, M. Wireless sensor networks: scheduling for measurement and data reporting. *IEEE Transactions on Aerospace and Electronic Systems*, 42, 2006.

[16] K. Ko Robertazzi and Thomas G. Scheduling in an environment of multiple job submissions, Mar. 2002 2002.

[17] Thomas G. Robertazzi. Processor equivalence for daisy chain load sharing processors. *IEEE Transactions on Aerospace and Electronic Systems*, 29(4):1216–1221, 1993.

[18] Thomas G. Robertazzi. Ten reasons to use divisible load theory. *Computer*, 2003.

[19] De Turck-F. Dhoedt B. Demeester P. Thysebaert, P. Using divisible load theory to dimension optical transport networks for computational grids. In *Optical Fiber Communication Conference, 2005. Technical Digest. OFC/NFOEC*, volume 2, 2005.

[20] B. Wong Han Min, Veeravalli. Aligning biological sequences on distributed bus networks: a divisible load scheduling approach. *IEEE Transactions on Information Technology in Biomedicine*, 9(4):489–501, 2005.

[21] C.C Ko Xiaolin Li, V. Bharadwaj. Divisible load scheduling on single-level tree networks with buffers constraints. *IEEE Transactions on Aerospace and Electronic Systems*, 36(4):1298–1308, 2000.

[22] H. Yang Yang, Casanova. Umr: a multi-round algorithm for scheduling divisible workloads. In *Parallel and Distributed Processing Symposium*, 2003.