Stony Brook University

Analysis of Load Distribution Strategies for Signature Search and Join Operation in Distributed Computing Systems

A Dissertation Presented

by

Yuntai Kyong

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

May 2010

Stony Brook University

The Graduate School

Yuntai Kyong

We, the dissertation committee for the above candidate for the
Doctor of Philosophy degree,
hereby recommend acceptance of this dissertation.

Thomas G. Robertazzi, Dissertation Advisor
Professor, Department of Electrical & Computer Engineering

Sangjin Hong, Chairperson of Defense
Associate Professor, Department of Electrical & Computer Engineering

Alex Doboli
Associate Professor, Department of Electrical & Computer Engineering

Esther Arkin
Professor, Department of Applied Mathematics & Statistics

The dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Analysis of Load Distribution Strategies for Signature Search and Join Operation in Distributed Computing Systems

by

Yuntai Kyong

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

2010

Divisible Load Theory(DLT) provides an optimal criterion for scheduling computational load in distributed computing system with commumincaiton cost. Divisible load is characterized by its infinite divisibliity, where computational load can be arbitrarly partitioned and there is no dependecny between partitioned load. Such a model is applicable where a large amount of data with little or no locality needs to be processed. Such data is readily found in the research community in simulating and processing scientific experiment such as particle accelearators, astronomical visual data and genome data. The arbitrary divisibility property of the load and the optimum criterion developed in the DLT literature leads to tractable algebraic solution for a given network architecture and load distribution policy. In the DLT literature,

the divistible load is characterized by its computation and comunication intensity, which specify the speed of computing and the cost of transferring the load. The solution for optimal load distribution exists for many variety of network architectures with scheduling policies. In the second chapter, a method of deriving the computing capacity of a computing cluster consisted of a large number of computers using DLT is examined. Instead of assuming that the whole load is available to the computing cluster, we consider a case where multiple types of loads are streamed in a stationary manner to the cluster with specific incoming rate for each load. In this work, we assume a bus network architecture where computing nodes are connected to a single dispatcher using a shared communication channel. In the third chapter, the closed form solution is derived for the expected search time of kth signature in the data set. This work is the extension of [1], where the expected time of single signature search is obtained. The work is extended to consider the search time of kth signature in the divisible load with arbitrary statisitcs of the location of the signature. The work also includes a method to speed up the signature search time by rearranging the load before distributing to the processor. The operation model assumed here is the linear search of signature in the load but it opens up a new aveune for further research issues when other operation models are considered. Another typical operation is the relational operation between records when a large amount of structured records is modeled as divisible load. In the forth chapter, analysis of distributed of join operation using divisible load theory is presented. The performance of Distributed Sort-merge join and Inner-Loop join operations are analyzed with with DLT and the theoretical maximum number of processors that can be utilized is derived. The analysis allows both

performance prediction and the development of efficient database algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Divisible load is a type of load for computation which can be arbitrarily partitioned and processed among a number of processors and links. There is no dependency among partitioned loads. Notably, the computational time and communication time remain proportional to the amount of partitioned load although non-linear computation time is considered in [2]. The paradigm of divisible load theory, as presented in [3] and [4], is well suited to numerous applications where large amount of independent data is processed in multiple computing processors. Such data models are easily found in large science project such as particle accelerators such as the Large Hadron Collider (LHC) [5] of European Organization for Nuclear Research(CERN). The amount of data collected during experiments is about 700 megabytes per second (MB/s). This huge amount of data is distributed to multiple computing site with multiple tiers, where chunks of data are distributed to multiple sites that are geographically dispersed with non-negligible communication delay. This approach is called Grid Computing and divisible load theory is well suited to this as it take

communication cost into account as well as computation time. The software of Grid computing, called middleware, is now used extensively for various scientific research projects such as analyzing extraterrestrial signal of Search from Extraterrestrial Intelligence(SETI) [6], where radio telescope data is downloaded to personal computers for which computing time is volunteered. Such data does not exhibit interdependence between partitioned data and is processed by same the operation so it can be well modeled as divisible load.

Without a constraint to match a specific processor to a specific load because of the divisibility property, the main focus in this research area has been how to distribute load in a way to finish computation of total load as fast as possible. The optimal distribution criterion is rigorously proved and presented in [7]. The criterion is for a minimum time solution all the processors must stop computing at the same instant. Under this criterion, a large amount of work has been published for various architectures of networks with different scheduling strategies. In [8] closed form solutions are derived for bus and tree networks. Star networks are also examined in [9]. As an extension to single level tree network, in [10], the problem of sequencing of operations and its closed form solutions is presented on arbitrary processor trees. In [11], applying divisible load theory on hypercube architectures are considered.

Scheduling strategies with multi-installment have been proposed in order to make computation start earlier than in the single-installment case. In [12], multi-installment of load distribution is studied in tree network. Also, a multiple job environment is discussed in [13] and [14], where a bus network architecture is assumed.

In the early phase of research in this area the focus are on the single-source

scheduling case. However, in a typical grid system, there are numerous computing sites that submit their computing load to grid system. In [15] and [16], the scheduling problem of divisible load with multiple sources is considered. In [16], the closed form of the solution of scheduling divisible load from multiple sources are derived.

The study of divisible load often sheds light on the architectural issue of distributed computing system. When sequencing is not allowed for a given network architecture optimal arrangements of processors are required for the optimal solution. Also, as the theory can incorporate non-negligible communication cost, the location of the computing process is important in minimizing overall finish time. In [17], optimal sequencing and arrangement is examined for the computing process in single-level tree networks. In [18], the load sequencing problem is considered in the context of minimizing monetary cost.

With non-negligible communication cost, adding processors to the system does not linearly translate into the enhancement of performance, i.e shorter finish time of the computing operation. Performance limits for distributed computing system are derived in [19]. In [20], the performance limits is studied for a two dimensional network.

Apart from deriving theoretical performance bound and optimal solution, system constraints such as start-up delay, limited size of buffers and release time have been studied as well. In [21], the effect of start-up delays in bus network is discussed. Reference [22] deals with the case with arbitrarily processors release time, while [23] studies scheduling strategies on multi-level tree networks with buffer constraints.

Divisible load theory found good applications in wireless sensor network, where

data generated by sensor nodes need to be collected to central node fur further processing. Sensor data is well modeled to divisible load as dependency between data from different sensors have little or no dependency. In [8], the load sharing problem of network sensors in bus network architectures is studied.

In Chapter 2, the capacity of computing clusters is derived using divisible load theory. When large amounts of data is generated such as the rate of 700 megabytes per second from the LHC, the storage of data itself is a big challenge and initial data processing before analysis is required. Such data often needs to be processed in real-time to unburden storage requirements. However, processing such a large volume of data in one computing processor is often prohibitively costly. We examine a case where steady state data is arrives at a load dispatcher which is connected to bus network with other computing processors. In the DLT literature, it is usually assumed that whole amount of load to distribute is readily available. In this work, it is assumed that the size of load is not known beforehand but only their incoming rate is specified. Based on divisible load theory, the capacity of the computing clusters are derived for multiple types of load where each type of load has different incoming rate and computation intensity. That is, for the same size of loads of different type, the computation time is different for the same speed of the processor. The derived result can serve as a guideline of capacity planning of the computing cluster and the theoretical upper bound of the capacity. The solution is derived with a deterministic model without buffer constraints. The actual distribution mechanism and the effect of the stochastic nature of incoming load is left for future study.

In Chapter 3, the evaluation of search time of multiple signatures in flat file

database is presented. Large sets of structured data such as database records can be modeled as divisible load when the size of records are small enough compared to the total size of records under consideration. Such database system has been used in aerospace application. In [24], database system is considered as the heart of integrated avionics systems and [25] discussed real time database system for avionics. Ko and Robertazzi in [1] apply the divisible load theory analytically to find a distinct pattern or signature in flat file records for the first time. In this work, flat file records are assumed as it is a natural choice for initial data processing [26]. In their work, closed form solution for the search time of a single signature is derived for both single level tree network and linear network. Also, the equivalent search time for multiple signature is derived. In this work uniform distribution of the signature in the dataset is assumed as it is feasible model of distribution of signature in large database as discussed in [27]. The third chapter of this thesis is the extension of this work in two ways. First, a closed form solution for search time of kth signature among multiple signature is derived using order statistics [28] for arbitrary probability distribution as to the location of the signatures. Secondly, a method of to speed up the signature search time when the distribution is not uniform is presented. The procedure is developed and simulated and, with a highly skewed distribution, it is shown that performance improvement can be significant.

In Chapter 4, an analysis of distributed join operations using DLT is presented. In this chapter, DLT is applied to two popular join operations, Sort-Merge Join and Hash Join, to find the optimal speed up for a given number of distributed processors and its performance bound.

In Chapter 5, the dispatch strategies in multiple robot environment is examined for the delivery type application. We employed a path-based model where the mobile robots follow the predefined path between locations for the delivery requests. A strategy based on generating minimum additional distance for each request for robot movement is proposed.

Finally, Chapter 6 concludes the thesis and discussion on the future works are presented.

Chapter 2

Capacity Analysis of Computing Cluster with Divisible Load

2.1 Introduction

We examine how the capacity of a cluster of processors can be obtained using Divisible Load Theory(DLT) when the different types loads, differentiated with different incoming rate and computational instantly, are streamed to the cluster. The calculated capacity can be utilized as a policy to decide when a new stream of load can be accepted to the cluster. Also, it will serve as a theoretical bound of the total capacity of the cluster We also present load distribution algorithm with DLT criterion and discuss the effect of the stochastic nature of incoming rate of load.

2.1.1 System Model

We consider a case when multiple sites submit to a computing cluster over network large amount of divisible load. We assume that the load is generated with specific deterministic rate and transfered to the computing cluster.

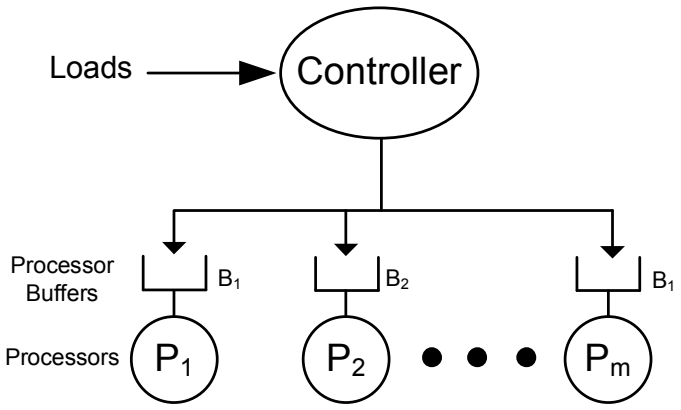


Figure 2-1: System Model

We further assume that basic information unit is a load contained in a fixed small size packet. In DLT literature, the total amount of load is known and set to 1 and the fraction of load distributed to each processor is calculated with the objective of minimizing total processing time. It is found that in order to minimize the finish time, the load has to be distributed to processors in a way that they stop computing at the same time. The following list is the summary of the system model.

- Information Unit: packet
- Packet size is small and constant
- Unknown total amount of load

- m processors, P_1,P_2,...,P_m, with inverse computing speed w_1, w_2,...,w_m with having dimension $\frac{sec}{packet}$
- Bus network with inverse speed z ($\frac{sec}{packet}$)
- Controller maintains a separate buffer for different loads.
- n types of loads arrive to controller ($L_1, \ldots, L_n$)
- Each load is specified with R_i, T_{cp}^i and common T_{cm}.
  - R_i is the inverse of incoming rate with dimension $\frac{sec}{packet}$
  - T_{cp}^i specifies the computational intensity of L_i
  - T_{cm} specifies the communication intensity of loads
- Local buffer for each process B_j for P_j, $j = 1, \ldots, m$
- The capacity of each buffer is given as the number of packets it can retain

2.2 Scheduling Loads with Unknown Size

2.2.1 Homogeneous Processing Speed with One Load

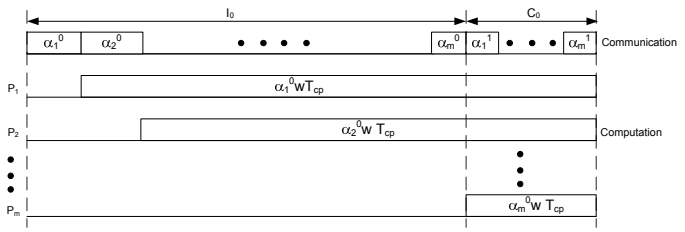


Figure 2-2: Initial Load Distribution for Homogeneous Processors

In this section, we consider a case when the speed of the processors are all equal and there is only one type of load specified with its inverse incoming rate R and its traffic intensity T_{cp}.

For an load with R and T_{cp}, the speed of bus should be able to support incoming rate $\frac{1}{R}$. For an arbitrary size of load γ, incoming rate $1/R$ decides the total time, T, to receive it.

$$T = \gamma R$$

The time T' to distribute the load over the bus network is,

$$T' = z\gamma T_{cm}$$

Therefore, in order for the bust network to support incoming rate $1/R$, $T' \leq T$,

$$T' = z\gamma T_{cm} \leq \gamma R = T$$

$$R \geq zT_{cm} \tag{2.1}$$

In Fig. 2-2, the load arrived during an arbitrary length of period I is first stored in the local buffer of each processor and its computation starts at the same time at the end of I. The incoming load is dispatched to each processor, in a way that the computation of the fraction of the load arrived during I by each processor stops at the same time following the optimally criterion of Divisible Load Theory. In the figure C denotes the computation time of the load. As (2.1) is satisfied, the load

can be distributed to the processor without being buffered in controller. However the distributed fraction has to be stored in the local buffer, B_i, of each processor before being computed.

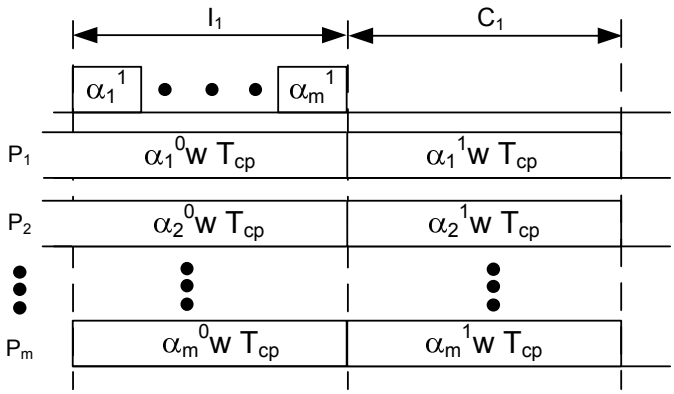


Figure 2-3: Load Distribution during I_1

Since the rate of load distribution during I is same as the incoming rate, provided that (2.1) is satisfied,

$$\frac{\alpha}{I} = \frac{1}{R}, \tag{2.2}$$

where α denotes total amount of load arrived and distributed during the interval I. When the speed of processors are all equal, the load needs to be equally distributed for each processor to compute its fraction during the same amount of time. Therefore,

$$C = \frac{\alpha}{m} w T_{cp} \tag{2.3}$$

From (2.2) and (2.3),

$$C = \frac{w T_{cp}}{mR} I \tag{2.4}$$

When $\frac{wT_{cp}}{mR} = 1$, $C = I$. In this case, the incoming rate of load exactly matches

computation rate of load. The load can be scheduled optimally in a sense that when the total amount of load is known and scheduled the computation is finished at the earliest time.

When $\frac{wT_{cp}}{mR} > 1$, $C > I$, which is the case when the aggregate computation rate is less than the incoming rate of load. During C, the amount load arrived is $\alpha * \frac{C}{I} = \alpha * \frac{wT_{cp}}{mR}$. In this case, the amount of load that need to be stored by local buffer keeps increasing. With a finite size of buffer, this load cannot be scheduled.

When $\frac{wT_{cp}}{mR} < 1$, $C < I$, which is the case when the aggregate rate of computation exceeds the incoming rate of load. In this case, the computational resource is available and the incoming rate of load can be increased up to the point when $\frac{wT_{cp}}{mR} = 1$.

In order for a load specified with $\{R, T_{cp}\}$ to be scheduled, the following condition should be satisfied,

$$\frac{T_{cp}}{R} \leq \frac{m}{w} \tag{2.5}$$

	$\frac{wT_{cp}}{mR}$	Schedulability	Utilization
Case I	$= 1$	Yes	100%
Case II	< 1	Yes	$< 100\%$
Case III	> 1	No	$\cdot$

Table 2.1: Schedulability and Optimality of Loads

TABLE 2.1 summarizes the condition of of schedulability and utilization depending on the value of $\frac{wT_{cp}}{mR}$.

Calculation of I, C and α_j

Fig. 2-4 describes the periodic scheduling of a single type of load on the processors with same speed. With the same processing speed, the batch is equally distributed

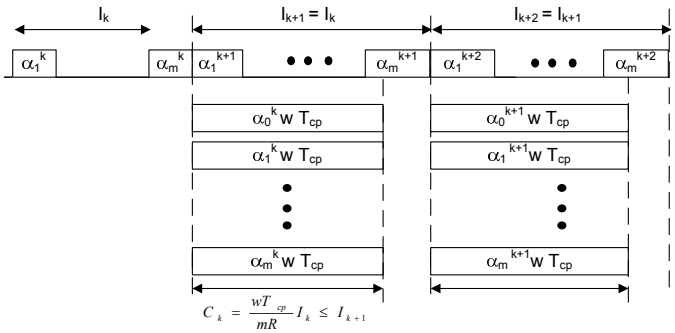


Figure 2-4: Load Distribution for I_i, $i \geq 2$

to each processor and for each interval the amount of load to be processed remains same.

$$\begin{aligned} \alpha_j &= \frac{I}{mR},\ j = 1, \ldots, m,\ i \geq 1 \\ C &= \frac{wT_{cp}}{mR} I \end{aligned} \tag{2.6}$$

I can be chosen arbitrarily but bounded by the size of local buffer. The load arrived during I and distributed to P_j cannot exceed the size of the local buffer, B_j. With homogeneous processor case,

$$\frac{I}{mR} \leq B_j,\ 1 \leq j \leq m \tag{2.7}$$

Also increasing I also increases the response time of the system.

2.2.2 Non-Homogeneous Processing Speed and One Load

In this subsection, we examine when the speed of processors are not same. By equating total load distributed during and interval I_k to the load arrived in the same period,

$$\frac{\alpha}{I} = \frac{1}{R}, \tag{2.8}$$

where $\alpha = \sum_{j=1}^{m} \alpha_j$.

In order for all the fractions of the loads arrived during I to be computed during a same duration C on each processor P_j,

$$\alpha_j w_j T_{cp} = C,\ j = 1, \ldots, m \tag{2.9}$$

By summing up the all the fractions,

$$\alpha = \sum_{j=1}^{m} \alpha_j = \sum_{j=1}^{m} \frac{C}{w_j T_{cp}} = C(\frac{1}{w_1 T_{cp}} + \frac{1}{w_2 T_{cp}} + \cdots + \frac{1}{w_m T_{cp}})$$

Using (2.8),

$$\frac{I}{R} = \frac{C}{T_{cp}}(\frac{1}{w_1} + \frac{1}{w_2} + \cdots + \frac{1}{w_m}) \tag{2.10}$$

$$C = \frac{T_{cp}}{R}(\frac{1}{w_1} + \frac{1}{w_2} + \cdots + \frac{1}{w_m})^{-1} I = \frac{T_{cp}}{RW} \tag{2.11}$$

where we denote W for the sum of inverse speed of each processor:

$$W = \sum_{j=1}^{m} \frac{1}{w_j} \tag{2.12}$$

When $\frac{T_{cp}}{RW} \leq 1$, $C \leq I$, and the load can be scheduled.

When the speed of processors are all same, the condition is reduced to the previous case:

$$C = \frac{wT_{cp}}{mR} I$$

Calculation of $\alpha_j, 1 \leq j \leq m$

We set $I = C_0$ and using (2.9) and (2.11), α_j^k is obtained as,

$$\alpha_j = \frac{\frac{1}{w_j}}{W} \frac{I}{T_{cp}R},\ j = 1, \ldots, m,$$

and C follows from (2.11).

I can be arbitrary chosen but bounded by the size of local buffers.

$$\alpha_j = \frac{1/w_j}{W} \frac{I}{T_{cp}R} \leq B_j,\ j = 1, \ldots, m \tag{2.13}$$

From this,

$$I \leq B_j \frac{W}{1/w_j} T_{cp}R \tag{2.14}$$

2.2.3 Schedulability of Multiple Divisible Loads

So far, we have considered the case where there is only one load. In this subsection, we examine the schedulability of multiple loads if their respective arrival rate, R_i, and, their computation intensities, T_{cp}^i, are all different. We also assume that the processing speed of processors are all different.

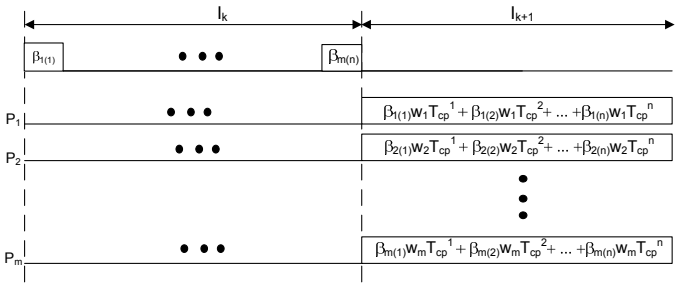


Figure 2-5: Scheduling Multiple Loads

With multiple loads (R_i, T_{cp}^i), the aggregate incoming rate needs to be supported by the communication bus,

$$R_1 + R_2 + \cdots + R_n \geq zT_{cm}, \tag{2.15}$$

when there are n loads to be scheduled as shown in Fig. 2.2.3.

Given that this condition is satisfied, for an interval I, the amount of portion from each load is obtained as,

$$\alpha^1 R_1 = \alpha^2 R_2 =, \ \ldots, \ = \alpha^n R_n = I, \tag{2.16}$$

where α^i indicates the amount of portion to be distributed during I from the load i, L_i.

Let α_j^i be the amount of load distributed to P_j from L_i in the interval I. Then,

$$\alpha^i = \sum_{j=1}^{m} \alpha_j^i. \tag{2.17}$$

If we let C_j be the computation time of the load arrived during I of P_j, we have

$$\begin{aligned} C_1 &= \sum_{i=1}^{n} \alpha_1^i w_1 T_{cp}^i \\ C_2 &= \sum_{i=1}^{n} \alpha_2^i w_2 T_{cp}^i \\ \cdots & \\ C_m &= \sum_{i=1}^{n} \alpha_m^i w_m T_{cp}^i. \end{aligned}$$

For optimal distribution, we set $C = C_1 = C_2 = \cdots = C_m$.

Also, from (2.16) and (2.17),

$$I = \sum_{j=1}^{m} \alpha_j^i R_i, \ i = 1, .., n$$

One optimal distribution can be found by forcing each load to be computed during same amount of time. For the load ith load,

$$\begin{aligned} \alpha_1^i w_1 T_{cp}^i &= C^i \\ \alpha_2^i w_2 T_{cp}^i &= C^i \\ &\cdots \\ \alpha_m^i w_m T_{cp}^i &= C^i, \end{aligned}$$

where C^i denotes the amount of time during which the fraction from the first load distributed to P_j is computed. Using above equations, we can obtain the fraction for each processor from the ith load,

$$\alpha_j^i = \frac{C^i}{w_j T_{cp}^i} \tag{2.18}$$

Summing up the fraction of L_i distributed to each processor,

$$\alpha^i = \sum_{j=1}^{m} \alpha_j^i = \frac{C^i}{T_{cp}^i} \sum_{j=1}^{m} \frac{1}{w_j} \tag{2.19}$$

Since $\alpha^i = \frac{I}{R_i}$,

$$C^i = \frac{T_{cp}^i}{W R_i} I, \tag{2.20}$$

where $W = \sum_{i=1}^{m} \frac{1}{w_i}$.

When the time of computation of each load is same for all the processors as we enforced, the total computation time of the load arriving during I on every processor

is given as,

$$C = \sum_{i=1}^{n} C^i, \tag{2.21}$$

Using (2.20),

$$C = \sum_{i=1}^{n} C^i = \frac{I}{W} \sum_{i=1}^{n} \frac{T_{cp}^i}{R_i} \tag{2.22}$$

From this we derive the schedulability condition for multiple load cases,

$$\frac{C}{I} = \sum_{i=1}^{n} \frac{T_{cp}^i}{WR_i} \tag{2.23}$$

This collection of loads can be scheduled when,

$$\sum_{i=1}^{n} \frac{T_{cp}^i}{WR_i} \leq 1 \tag{2.24}$$

When the computation intensity and the incoming rate are all same for n loads, the previous equation is reduced to

$$\begin{aligned} n\frac{T_{cp}}{WR_i} &\leq 1 \\ n \leq \frac{WR}{T_{cp}} &= \frac{mR}{wT_{cp}} \end{aligned} \tag{2.25}$$

where we can see that the maximum number of loads that the system can support is $\frac{mR}{wT_{cp}}$.

When there were $(n-1)$ loads scheduled, the following two conditions should have

been satisfied to one more load to be scheduled:

$$\sum_{i=1}^{n-1} R_i \leq zT_{cm} \tag{2.26}$$

and,

$$\sum_{i=1}^{n-1} \frac{T_{cp}^i}{WR_i} \leq 1 \tag{2.27}$$

With a new load with R_n and T_{cp}^n, the following conditions should be satisfied,

$$\begin{aligned} \sum_{i=1}^{n-1} R_i + R_n &\leq zT_{cm} \\ R_n &\leq zT_{cm} - \sum_{i=1}^{n-1} R_i \end{aligned}$$

and,

$$\begin{aligned} (\sum_{i=1}^{n-1} \frac{T_{cp}^i}{R_i} + \frac{T_{cp}^n}{R_n})W^{-1} &\leq 1 \\ \sum_{i=1}^{n-1} \frac{T_{cp}^i}{R_i} + \frac{T_{cp}^n}{R_n} &\leq W \\ \frac{T_{cp}^n}{R_n} &\leq W - \sum_{i=1}^{n-1} \frac{T_{cp}^i}{R_i} \end{aligned}$$

Load Distribution

From (2.18) and (2.20),

$$\alpha_j^i = \frac{I}{R_i} \frac{\frac{1}{w_j}}{W} \tag{2.28}$$

I is upper bounded by the size of local buffer of each processor. For P_j, from

$$\sum_{i=1}^{n} \alpha_j^i \leq B_j \tag{2.29}$$

$$\begin{aligned} \sum_{i=1}^{n} \frac{I}{R_i} \frac{1/w_j}{W} &\leq B_j \\ I &\leq \frac{B_j}{1/w_j} \frac{W}{\sum_{i=1}^{n} 1/R_i} \end{aligned} \tag{2.30}$$

which can be plugged into (2.28) to get actual amount of load fraction for each L_i and P_j.

2.3 Load Distribution Mechanism

The load distribution calculated with (2.28) only specifies how much load should be distributed from which load and to which processor during I, but it does not specify when the fraction should be distributed to the processors.

Let us define the ratio of load distributed to P_j to the total load from L_i during I_k, where L_i denotes the ith load.

$$f_j^i = \frac{\alpha_j^i}{\sum_{j=1}^{m} \alpha_j^i} = \frac{\alpha_j^i}{\alpha^i} = \frac{1/w_j}{W} \tag{2.31}$$

Since this ratio is calculated with a fixed but an arbitrary length of I, this ratio remain constant for any subinterval within I'. During I' within I, the amount of load

to be distributed from L_i to P_j is,

$$\beta_j^i = f_j^i * \beta^i, \tag{2.32}$$

if β^i is the amount of load from L_i during I' and β_j^i is the amount of its fraction distributed to P_j.

This property holds any subinterval in I. If we let I^l be subinterval within I and $1 \leq l \leq L$, where L is the number of subintervals in I, $I = \sum_{l=1}^{L} I^l$. When deterministic arrival rate is assumed, at the end of each subinterval, the amount of data from each load can be obtained. From

$$\frac{1}{R_i} = \frac{\alpha^{i(l)}}{I^l} = \frac{\alpha^i}{I}, \tag{2.33}$$

We have

$$\alpha^{i(l)} = \frac{I^l}{I}\alpha^i, \ i = 1, \ldots, n. \tag{2.34}$$

If the distribution is made according to the ratio f_j^i, at the end of I^l,

$$\alpha_j^{i(l)} = f_j^i \alpha^{i(l)} \tag{2.35}$$

When the load is distributed according to the radios f_j^i at the end of each subin-

terval,

$$\alpha^i = \sum_{j=1}^{m}\sum_{l=1}^{L}\alpha^{i(l)} = \sum_{j=1}^{m}\sum_{l=1}^{L} f_j^i \alpha^{i(l)} \tag{2.36}$$

$$= \sum_{j=1}^{m} f_j^i \sum_{l=1}^{L}\alpha^{i(l)} \tag{2.37}$$

$$= \sum_{j=1}^{m} f_j^i \alpha^i = \alpha^i. \tag{2.38}$$

where we use $\sum_{j=1}^{m} f_j^i = 1$ and (2.35).

One approach for distribution is split each packet into j pieces and distribute to j processors according to f_j^i, which involves additional processing overhead and diminish effective communication speed z due to the overhead of protocol stacks also it is not plausible to separate small amount of data. We assume in this work, individual packet cannot be divided. That is, the packet is basic unit of divisible load.

2.3.1 DLT based dispatching policy

In the previous subsection, with deterministic incoming rate of each load, the distribution ratio f_j^i remains constant and load arrived during any duration can be distributed to processors according to these ratios in order to obtain optimal distribution. In this subsection, instead of considering load during a subinterval, which involves additional buffering of incoming load, we examine a policy for dispatching an incoming packet instantaneously to processors based on DLT criterion. In our policy, each packet, when it arrives to a controller, is instantly dispatched to a processor that balance the current load distribution ratio, f_j^i, in a greedy fashion. Our approach is different from

the traditional dispatching policy of scheduling server farm as it does not depend on the state of buffer of each processor or the processor is not decided randomly.

I	Current Interval Period
f[i]	Pre-calculated distribution ratio
D[i,j]	Record the number of packets sent to P_j from L_i
$\alpha[i]$	Count the number of arrived packets in the current period

Table 2.2: Operation Table for Packet Dispatching Algorithm

TABLE II describes the operation tables that the controller maintains to be used in the distribution mechanism. At the beginning of every interval I, $D[i,j]$ and $\alpha[i]$ are reset to zero. For every packet arrived from L_i, $\alpha[i]$ is increased by one. When the mechanism decides to send the packet to P_j, $D[i,j]$ is incremented. $f[j]$ is the recalculated distribution ratio for each P_j. For now, I remains constant for every interval.

Algorithm 1: Load Distribution Mechanism

Input: Packet p Arrived from L_i

$\alpha[i] \leftarrow \alpha[i] + 1$

For each $j \in [1, .., M]$, $f_j'^i \leftarrow \frac{D[i,j]+1}{\alpha[i]}$

$j = \arg_j \ min(|f_j^i - f_j'^i|)$

Send the packet to P_j

$D[i,j] \leftarrow D[i,j] + 1$

Algorithm 1 is invoked whenever an packet is arrived to the controller at instant t. Whenever it is invoked, the controller decides to which processor the packet should be dispatched in a way that the actual load distribution is as close as possible to the pre-calculated load distribution ratio $f_{j(i)}^k$. Algorithm 1 choose a processor P_j

when the packet is dispatched to it, the difference between actual and theoretical distribution ratio is minimized with P_j. When the j is chosen as to minimize the difference between f_j^i and $f_j'^i$, if there is tie, j is arbitrarily chosen.

2.4 Effect of Non-deterministic Incoming Rate

DLT was developed with deterministic model. This remains valid when the total amount of load is known and the rate of computation remain constant. However, when considering multiple incoming streams of load, the arrival rate is hard to stay deterministic especially when they are submitted using network. This section we examine how the non-deterministic nature of incoming rate R_i affects the total capacity of the system. We choose I sufficiently large so that R_i remains constant within an interval.

$$\tilde{R}_i = R_i(1 + \delta_i). \tag{2.39}$$

We also assume that $\sum_k \delta_i^k \approx 0$ over the sufficiently large number of intervals and $\delta_i^k \ll 1$.

When the load is distributed according to the prescribed incoming rate, from (2.31),

$$\alpha_j^i = \alpha_*^i f_j^i = \frac{I}{R_i} * f_j^i \tag{2.40}$$

Actual computation time $\tilde{C_j}$ for P_j is obtained as,

$$\tilde{C_j} = \sum_{i=1}^{n} \frac{I}{\tilde{R_i}} \frac{T_{cp}^i}{W} \tag{2.41}$$

$$= \frac{I}{W} \sum_{i=1}^{n} \frac{T_{cp}^i}{R_i(1+\delta_i)} \tag{2.42}$$

As it can be seen, the computation time remains same for all the processors as R_i varies. It is because the variation of incoming rate equally affects the computation time of each processor. $\tilde{C_j} = \tilde{C}$.

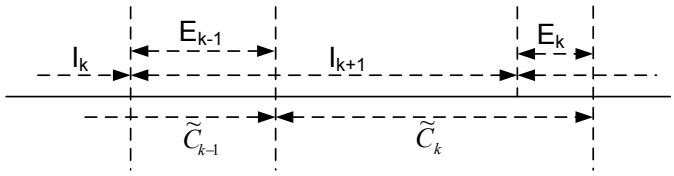


Figure 2-6: Timing Diagram For for Computation Time with Non-Deterministic Incoming Rate

In Fig. 6, we subscript the intervals. I_{k+1} is the next interval of I_k. Also $\tilde{C_k}$ denotes the computation time for the load arrived during I_k. When the computation for the load arrived ruing I_k is not finished during the next interval I_{k+1}, the computation time is extended into the next period. We denote this excessive computation time as E_k. E_k depends on the previous excessive computation time E_{k-1} and current computation time $\tilde{C_k}$. Therefore, we have

$$E_k = E_{k-1} + \tilde{C_k} - I_{k+1} \tag{2.43}$$

Assume that δ_i^k does not depend on k. If we let $X_k = \tilde{C}_k - I$. Then,

$$X_k = \left(\frac{1}{W} \sum_{i=1}^{n} \frac{T_{cp}^i}{R_i(1+\delta_i)} - 1 \right) I \tag{2.44}$$

$$X_k = \left(\frac{Y}{W} - 1 \right) I \tag{2.45}$$

where

$$Y = \sum_{i=1}^{n} \frac{T_{cp}^i}{R_i(1+\delta_i)} \tag{2.46}$$

If we set $E_0 = 0$,

$$E_k = \sum_{i=1}^{k} X_i \tag{2.47}$$

Whenever $E_k <= 0$, the additional buffer is emptied and the process parasitically reset. Therefore, we can let subscript k indicate the number of period since the last time when the additional buffer is empty. We need to find the statistics of E_k which specifies the excessive buffer size that needs to be provided for a given δ_i. When the statistics of δ_i is known, that of X_i is also known. From this statistics, the statistics of $\max_k E_k$ can be obtained. We assume that maximum excessive time E is given for the system which is obtained from the size of local buffers, since the load that is calculated within excessive time, should be stored in the buffer.

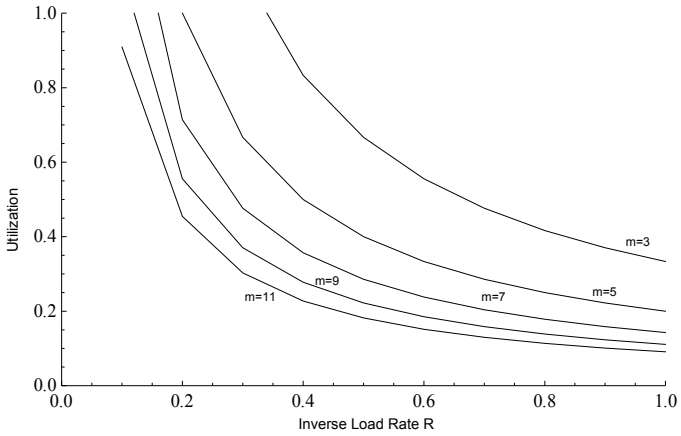


Figure 2-7: Utilization versus inverse incoming rate of load R. $T_{cp} = 1, w = 1$

2.5 Evaluation and Analysis

2.5.1 Utilization of Homogeneous System

Fig. 7 shows the relationship between inverse rate of incoming load and the utilization of the system for various number of processors with fixed traffic intensity T_{cp} and inverse computing speed, w, of the processors. As the number of processors increases, the system can accommodate more incoming load.

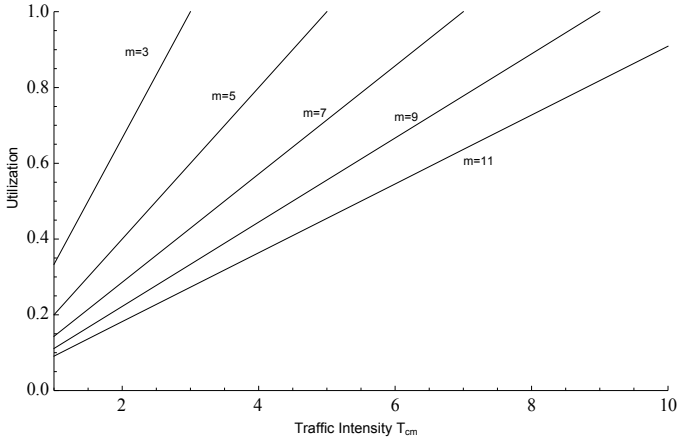


Figure 2-8: Utilization versus Computation Intensity T_{cp}. $R = 0.1, w = 1$

Fig. 8 shows the linear relationship between the utilization of the system against the computation intensity(T_{cp}).

2.6 Conclusion and Future Work

The computation capacity of the computing clusters is derived for the bus network with communication delay. As the deterministic incoming rate of loads is assumed, actual capacity is lower than the theoretical capacity obtained. Analysis of the effect with stochastic arrival rate and effective distribution mechanism is remained as future work.

Chapter 3

Signature Search Time and Load Distribution Strategy for Minimizing Signature Search Time in Divisible Load

3.1 Introduction

A "signature" is a data pattern of interest in a large data record. Large sets of structured data such as database records can be modeled as divisible loads when the size of records are small enough compared to the total size of records under consideration. The search for signatures occurs in radar and sensor data processing. Such database system has been used in aerospace applications. In [24], database

systems are considered as the heart of integrated avionics systems and [25] discussed real time database system for avionics. Divisible load theory was introduced by [29] and [30] and, also independently in [31] as "large-grained parallelism". Ever since its introduction, divisible load theory has served as an effective modeling tool for data-intensive applications [7] [32] [3] and a large amount of work has been published for various network structures including linear networks [33], bus networks [34], single and multi level tree-networks [35] [36] [37], mesh networks [38] [39], hypercubes [40] and arbitrary graphs [41] with different constraints such as different release times, buffer constraints [42], communication start-up costs and time-varying network capacity. Also, various distribution strategies have been studied including multi-installment of load distribution [43]. In [44], [45] and [46], scheduling strategies without knowledge of network resources are examined. In [47] signature searching is investigated on bus networks experimentally. Ko and Robertazzi in [1] applied divisible load theory analytically to find a distinct pattern or signature in flat file records. In this work, flat file records are assumed as it is a natural choice for initial data processing [26]. A closed form solution for the average search time of a single signature is derived for both a single level tree network and a linear network. In [1] a uniform distribution of the signature in the dataset is assumed as it is a feasible model of the distribution of signatures in large database as discussed in [27].

This paper consists of two parts. First, a closed form solution for expected search time of the kth signature among multiple signature is derived using order statistics [28] for arbitrary probability distributions modeling the location of the signatures. Secondly, a method to speed up the expected signature search time when the distribution

of the signatures is not uniform is presented. The procedure is developed and simulated and it is shown that performance improvement can be significant when the signatures are highly concentrated in a certain region of the load.

3.2 System Model

3.2.1 Network Model

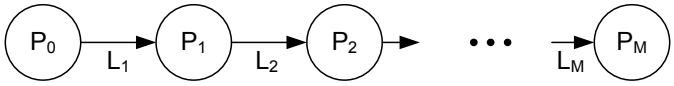


Figure 3-1: Model of Linear Daisy Chain Network with Communication Links.

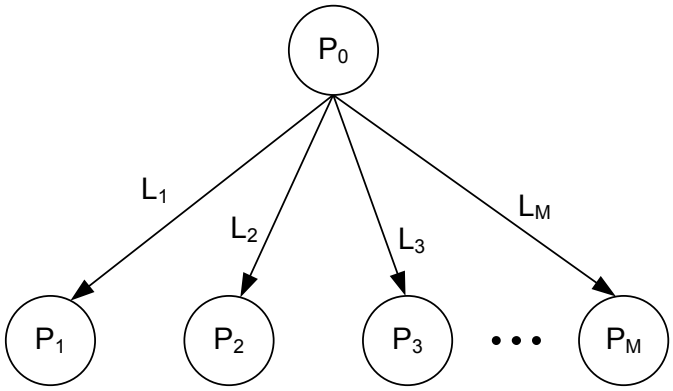


Figure 3-2: Model of Single Level Tree Network with Communication Links.

We consider signature search time on two network models using divisible load theory. Fig. 3-1 describes a linear daisy chain model and, in Fig. 3-2, a single level tree network model is described. In this second network model, one root processor, P_0, is connected to the rest of the processors via links. In both models, the load is originated from P_0. Communication links connected to P_j are shown as L_j. The notation used in this paper is summarized in TABLE 3.1. Here α_i is the optimal

α_i	Fraction of entire processing load assigned to ith processor
w_i	Constant that is inversely proportional to computation speed of ith processor
z_i	Constant that is inversely proportional to communication speed of ith link
T_{cp}	Computation intensity: time taken to process a unit load on ith processor when $w_i = 1$
T_{cm}	Communication intensity: time taken to communicate a unit load over link l_i when $z_i = 1$
S_i	Processing start time of ith processor
β^i	Fraction of load that is processed during S_i and S_{i+1}
$\|\beta^i\|$	The size of β^i
$l(\beta^i), u(\beta^i)$	The lower and the upper boundary of β_i
α^i_j	The fraction of load distributed to P_i taken from β^i
$\|\alpha^i_i\|$	The size of α^i_j
$l(\alpha^j_i), u(\alpha^j_i)$	The lower and the upper boundaries of α^j_i

Table 3.1: Summary of notation for Divisible Load Theory

fraction of load distributed to processors and links, calculated with divisible load theory using the rest of system parameters, w_i, z_i, T_{cp} and T_{cm}, which are given as constant values as system parameters. For the scheduling policy for a single level tree network, we consider only the single-installment case, where communication of the load to each processor takes place only once for each processor. We also assume that the originating node also computes the load. It is also assumed that a child can only begin processing its load fraction after it receives it in entirety. Furthermore, each processor is equipped with front-end processor for off-loading communication, so they can send the fractions of load to other processors while computing its own fraction. A closed-from solutions for the load distribution (α_i) and the expression for the finish time (makespan) appear in [7] and [1] and are reproduced in the next subsection.

3.2.2 Divisible Load Modeling

The solutions for the optimal load fraction to distribute to each processor and the starting time and the finish time of processing distributed load by each processor is obtained using recursive equations in [7].

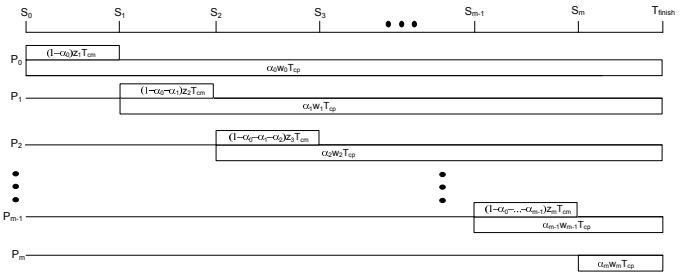


Figure 3-3: Timing diagram for load distribution on a linear daisy chain network

Fig. 3-3 and Fig. 3-4 show timing diagrams of load distribution for linear daisy

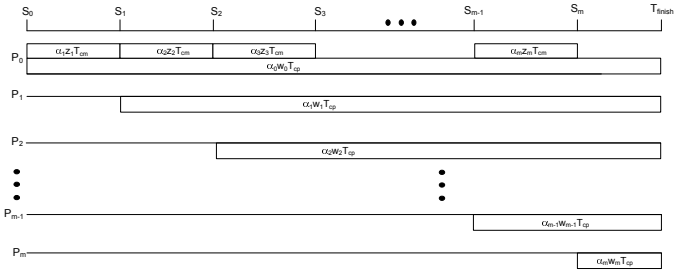


Figure 3-4: Timing diagram for load distribution on a single level tree network with single installment

chain network model and single level tree network model, respectively. It is shown that the processing time of the ith processor is given as $\alpha_i w_i T_{cp}$ and the load communication time is the amount of load to transfer multiplied by constant factor $z_i T_{cm}$. For example, in Fig. 3-4, the amount of time to transfer α_1 to P_1 is given as $\alpha_1 z_1 T_{cm}$. As the optimum criterion of divisible load theory states, the computation time has to be finished at the same instant by all processors, T_{finish}, by each processor. This criterion is intuitively reasoned in [7] - while distributing arbitrarily divisible loads, one should keep all the processors utilized until the last moment. If all processors do not stop at the same time, certainly the load can be transferred to idle processor to improve the solution. A rigorous proof of this optimum criterion for various network models is presented in [7]. With this constraint, M recursive equations can be written for each network model as,

$$\alpha_i w_i T_{cp} = (1 - \sum_{j=0}^{i} \alpha_j) z_{i+1} T_{cm} + \alpha_{i+1} w_{i+1} T_{cp}, \qquad i = 0, 1, \ldots, M-1, \tag{3.1}$$

for the linear daisy chain network model and

$$\alpha_i w_i T_{cp} = \alpha_i w_i T_{cp} = \alpha_{i+1} z_{i+1} T_{cm} + \alpha_{i+1} w_{i+1} T_{cp}, \quad i = 0, 1, \ldots, M-1, \tag{3.2}$$

for the single level tree network model. Each set of recursive equations, along with a normalization equation, form a system of $(M+1)$ linear equations with $(M+1)$ unknowns, $\{\alpha_0, \alpha_1, \ldots, \alpha_M\}$. The normalization equation is given as,

$$\sum_{i=0}^{M} \alpha_i = 1, \tag{3.3}$$

for both network models. The starting time, S_i, of ith processor, P_i, is determined

Network Model	S_0	S_m $(m = 1, \ldots, M)$
Linear Daisy Chain	0	$\sum_{j=0}^{m-1} \left[\left(1 - \sum_{k=0}^{j} \alpha_k \right) \cdot z_{j+1} T_{cm} \right]$
Single Level Tree	0	$\sum_{j=1}^{m} \alpha_j \cdot z_j T_{cm}$

Table 3.2: Time to Start Searching

by communication time of the load to the previous processors, from P_0 to P_{i-1} and P_i itself. The starting time for each network model is given in [1] and reproduced in TABLE 3.2.

3.3 Expected Time of Searching For Multiple Signatures

3.3.1 Problem Description

We define a set of random variables $\{\mathbf{X}_i\}$ that describes the positions of K signatures in the dataset with the normalized size 1. We assume that the positions of the signatures have a certain distribution,

$$\{\mathbf{X}_i\} \sim \mathbf{F}, \tag{3.4}$$

where $\mathbf{F}$ is joint CDF defined on $[0,1]^K$. We denote $\mathbf{Y}_k$ as a random variable describing the amount of time to find the kth signature. The objective of the first part of this paper is to find the expectation value of $\mathbf{Y}_k$.

3.3.2 Uniform Distribution with Single Installment

When $\{\mathbf{X}_i\}$ are independent and identically distributed random variables with uniform distribution, $\mathbf{X}_i \sim U(0,1),\ i \in [1,K]$. The position of the kth signature is given as $\mathbf{X}_{(k)}$, the kth order statistics of $\{\mathbf{X}_i\}$. The $E[\mathbf{Y}_k]$ can be expressed as,

$$E[\mathbf{Y}_k] = \sum_{m=0}^{M} E[\mathbf{Y}|A_m]\Pr(A_m), \tag{3.5}$$

where A_m denotes the event when the kth signature is found on P_m. For the single installment case, we have,

$$\begin{aligned} \Pr(A_0) &= \Pr(0 \leq \mathbf{X}_{(k)} \leq \alpha_0) \\ \Pr(A_m) &= \Pr(\sum_{j=0}^{m-1} \alpha_j \leq \mathbf{X}_{(k)} \leq \sum_{j=0}^{m} \alpha_j),\ m \in [1, M]. \end{aligned} \tag{3.6}$$

Given that the kth signature is found on P_m,

$$\begin{aligned} \mathbf{Y}_m | A_m &= g_m(\mathbf{X}_{(k)} | A_m) \\ &= (\mathbf{X}_{(k)} | A_m - \sum_{i=1}^{m-1} \alpha_i) w_m T_{cp} + S_m, \end{aligned} \tag{3.7}$$

where $g_m(\cdot)$ is the transformation function presented in [1], which takes the position of a signature as an argument and gives the signature search time depending on which processor the signature is found. Here, S_m denotes the starting time of P_m as described in TABLE 3.2 and $(\mathbf{X}_{(k)} | A_m - \sum_{i=1}^{m-1} \alpha_i)$ is the offset of the position of the kth signature from the beginning of load fraction distributed to P_m.

The kth order statistics of the uniform distribution follows the Beta distribution,

$$\mathbf{X}_{(k)} \sim \mathbf{B}(k, K + 1 - k). \tag{3.8}$$

If we denote $f_{k,K}(x)$ as its PDF, given that $\mathbf{X}_{(k)}$ is on P_m, the conditional distri-

bution of $\mathbf{X}_{(k)}$ is given as,

$$\begin{aligned}\mathbf{X}_{(k)}|A_m &\sim \frac{f_{k,K}(x)}{\int_{\gamma_{m-1}}^{\gamma_m} f_{k,K}(x)dx} \\ &= \frac{f_{k,K}(x)}{I_{\gamma_m}(k) - I_{\gamma_{m-1}}(k)},\ \gamma_{m-1} \leq \gamma_m,\end{aligned} \tag{3.9}$$

where the limits of the integration γ_{m-1} and γ_m are $\sum_{j=0}^{m-1} \alpha_j$ and $\sum_{j=0}^{m} \alpha_j$, respectively. We set $\gamma_{-1} = 0$ for convenience. $I_x(k)$ is the shorthand notation for $I_x(k, K+1-k)$, which is the CDF of Beta distribution, $f_{k,K}(x)$.

$$I_x(k, K+1-k) = \int_0^x f_{k,K}(\zeta)d\zeta \tag{3.10}$$

It follows $I_{\gamma_{-1}} = I_0 = 0$. Taking the expectation of (3.7),

$$E[\mathbf{Y}_m|A_m] = (E[\mathbf{X}_{(k)}|A_m] - \sum_{i=0}^{m-1} \alpha_i) w_m T_{cp} + S_m. \tag{3.11}$$

From (3.9),

$$E[\mathbf{X}_{(k)}|A_m] = \frac{\int_{\gamma_{m-1}}^{\gamma_m} x f_{k,K}(x)dx}{I_{\gamma_m(k)} - I_{\gamma_{m-1}}(k)}, \tag{3.12}$$

From (3.6),

$$\begin{aligned}\Pr(A_m) &= \int_{\gamma_{m-1}}^{\gamma_m} f_{k,K}(x)dx \\ &= I_{\gamma_m}(k) - I_{\gamma_{m-1}}(k).\end{aligned} \tag{3.13}$$

Using theses results into (3.5), we have an expression for the expected time for

finding kth signature out of K signatures:

$$E[\mathbf{Y}_k] = \sum_{m=0}^{M} \left(\left(\frac{\int_{\gamma_{m-1}}^{\gamma_m} x f_{k,K}(x) dx}{I_\gamma(k) - I_{\gamma_{m-1}}(k)} - \gamma_{m-1} \right) w_m T_{cp} + S_m \right) \\ * \left(I_{\gamma_m}(k) - I_{\gamma_{m-1}}(k) \right). \tag{3.14}$$

Single Signature Case

When there is only one signature, we have $K = k = 1$. We use the identity,

$$I_x(a,b) = \sum_{j=a}^{a+b-1} \frac{(a+b-1)!}{j!(a+b-1-j)!} x^j (1-x)^{a+b-1-j}. \tag{3.15}$$

When $a = k = 1$ and $b = K + 1 - k = 1$,

$$I_{\gamma_m}(1,1) = \gamma_m = \sum_{j=1}^{m} \alpha_j \tag{3.16}$$

Then, $I_{\gamma_m}(k) - I_{\gamma_{m-1}}(k) = \sum_{j=0}^{m} \alpha_j - \sum_{j=0}^{m-1} \alpha_j = \alpha_m$. Since $\mathbf{X}_{(1)}$ is uniformly distributed, when $K = 1$, $E[\mathbf{X}_{(1)}|A_m] = (\gamma_m + \gamma_{m-1})/2$. Also, $(\gamma_m + \gamma_{m-1})/2 - \gamma_m = (\gamma_m - \gamma_{m-1})/2 = \alpha_m/2$. Substituting these results into (3.7),

$$\begin{aligned} E[\mathbf{Y}_1] &= \sum_{m=0}^{M} \alpha_m \left(\frac{\alpha_m w_m T_{cp}}{2} + S_m \right) \\ &= \sum_{m=0}^{M} \alpha_m \left(\frac{\alpha_m w_m T_{cp} + 2S_m}{2} \right) \\ &= \sum_{m=0}^{M} \alpha_m \left(\frac{T_{finish} + S_m}{2} \right), \end{aligned} \tag{3.17}$$

where T_{finish} denotes the time when the computation finishes and is given as $T_{finish} = \alpha_m w_m T_{cp} + S_m$. The expected time of single signature search time is found as the

weighted average of midpoints of the processing times of each processor. This special case confirms the result presented in [1].

Time to find the last signature

The distribution of the position of the last signature is given as,

$$\mathbf{X}_{(K)} \sim \mathbf{B}(K, 1). \tag{3.18}$$

The standard form of the PDF of the Beta distribution is given as,

$$f(x; a, b) = \frac{1}{B(a, b)} x^{a-1}(1-x)^{b-1}, \tag{3.19}$$

where $B(a, b)$ is the Beta function with the parameters a and b. When $k = K, a = K, b = K + 1 - k = 1$, $f_{K,K}(x) = f(x; K, 1) = \frac{1}{B(K,1)} x^{K-1}$. Also, from (3.15),

$$I_x(K, 1) = x^K \tag{3.20}$$

Using these with (3.11),

$$\begin{aligned} E[\mathbf{X}_{(K)}|A_m] &= \int_{\gamma_{m-1}}^{\gamma_m} x \frac{f(x; K, 1)}{\gamma_m^N - \gamma_{m-1}^N} dx \\ &= \int_{\gamma_{m-1}}^{\gamma_m} x \frac{x^{K-1}}{B(K, 1)} \frac{1}{\gamma_m^N - \gamma_{m-1}^N} dx \\ &= \int_{\gamma_{m-1}}^{\gamma_m} \frac{x^K}{B(K, 1)} \frac{1}{\gamma_m^N - \gamma_{m-1}^N} dx \\ &= \frac{K}{K+1} \frac{\gamma_m^{K+1} - \gamma_{m-1}^{K+1}}{\gamma_m^K - \gamma_{m-1}^K}, \end{aligned} \tag{3.21}$$

where we use $B(K,1) = \frac{1}{K}$. With (3.5) and (3.20),

$$E[\mathbf{Y}_K] = \sum_{m=0}^{M} \left(\left(\frac{K}{K+1} \frac{\gamma_m^{K+1} - \gamma_{m-1}^{K+1}}{\gamma_m^K - \gamma_{m-1}^K} - \gamma_{m-1} \right) w_m T_{cp} + S_m \right) * (\gamma_m^K - \gamma_{m-1}^K). \tag{3.22}$$

In [1], the expected search time of the last signature is derived using the concept of an equivalent processor for the uniformly distributed case. Here, a closed form solution with a general distribution is derived.

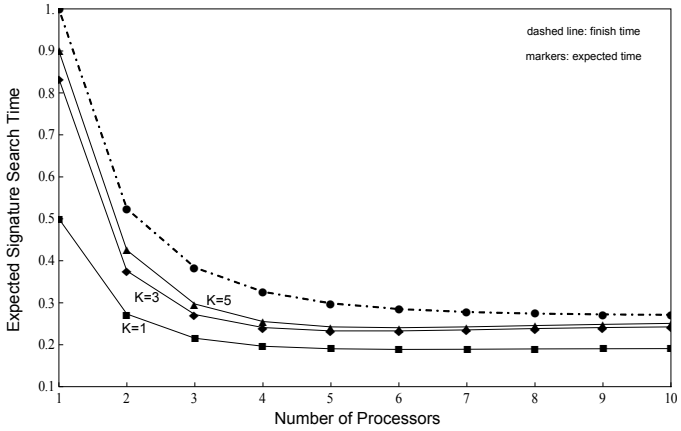


Figure 3-5: Linear daisy chain network: time to find the last signature versus number of processors, K signatures, $w = 1, z = 0.2, T_{cp} = 1, T_{cm} = 1$

Fig. 3-5 shows the search time for the last signature when there are K signatures. The dotted line at the top is the finish time when the processors do not stop processing after the last signature is found. When there are multiple signatures, adding more processors gives better speed up (compared to using only one processor) than when there is only one signature. However, as shown in the plot, as more processors are added the expected signature search time is slightly increasing after the speed-up is

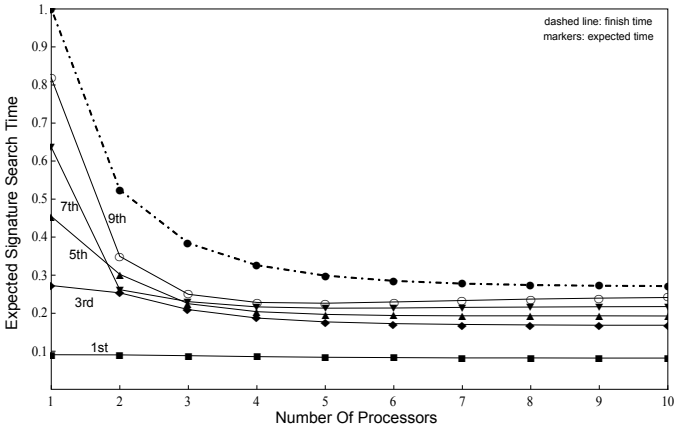


Figure 3-6: Linear daisy chain network: time to find the kth signature versus number of processors, 10 signatures, $w_i = 1, z_i = 0.2, T_{cp} = 1, T_{cm} = 1$

saturated. While the speed-up through parallel processing is not increased because of the communication overhead, when calculating the expected value of signature search time, the residual probability mass of finding signatures on the processors that are added after the saturation contribute to increasing the expectation value albeit in a negligible way.

Fig. 3-6 shows the search time for the kth signature when the number of signatures is fixed to $K = 10$. It can be observed that before the speed up saturates, the order of expected time to find the signature changes as shown in the figure.

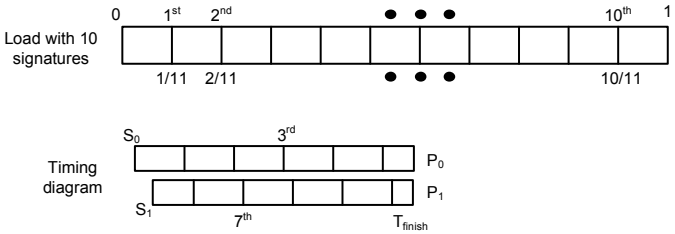


Figure 3-7: Load with uniformly distributed 10 signatures and mini timing diagram for two processors, $w = 1, z = 0.2, T_{cp} = 1, T_{cm} = 1$

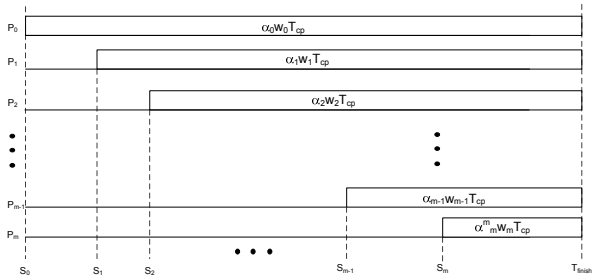
The reason for this can be explained with Fig. 3-7. In the figure, the upper part describes the normalized load with size 1 with 10 uniformly distributed signatures of

those expected locations are shown as vertical lines. With the same system parameters for generating the plot for Fig. 3-6, when there are two processors the load distribution is given as $\{0.52381, 0.47619\}$. The bottom part shows a mini timing diagram when the load is distributed with this proportion and the expected location of signatures. As shown in the timing diagram, the expected time to find 7th signature is smaller than the expected time of 3rd signature because of parallel processing. Although this diagram is not exact as it uses the expected positions, it explains why the order of expected search time may be switched as shown in Fig. 3-6. A similar effect is also pointed out in [1].

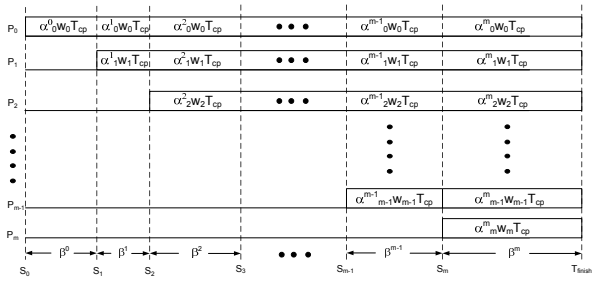
This all assumes a linear load is mapped continuously and sequentially into each load fraction for each processor. The signatures are not necessarily detected in the original order they appear in the original data flat file. In the case of signature searching if the original flat file is indexed by time, there may be some concern that signatures spanning two sides of a load partition are correctly detected. This can be handled by some overlapping of data fragments near partition boundaries. If this can be handled, then the original load could conceivably be multiplexed into each processing node sequentially and repetitively so that signatures are more likely to be detected in their original order if that is important.

3.4 Load Distribution Strategy For Minimizing Search Time

3.4.1 Load Classification based on Starting Time



(a) Sequential Load Distribution



(b) Load Distribution Based on Starting Time

Figure 3-8: Comparison of Load Distribution Scheme

Fig. 3-8(a) shows the timing diagram of the computation time of distributed load for a linear daisy chain network and a single level tree network. The load is distributed in sequential order as it has been assumed in the literature of divisible load theory, where the position of the fraction has not been considered. In sequential load distribution, once α_i, the fraction for P_i is determined, P_0 gets the first part of load and P_1 receives the next part of load. The ranges of the fractions inside the total

load are $[0, \alpha_0]$ and $[\alpha_0, \alpha_0 + \alpha_1]$, respectably, for P_0 and P_1.

However, when the likelihood of finding signatures are not uniform, by computing earlier the fractions of load with a higher likelihood of finding the signatures, the expected time of finding the signature will be shown to be reduced. Fig. 3-8(b) shows the classification of load based on their starting time of computation. Here β^i denotes the fraction of load which is computed between S_i and S_{i+1}. Since β^0 is processed at the earliest time, it should contain the fraction of load that has highest probability of containing signatures. The fraction, β^1, having the highest probability excluding β^0 is processed between S_1 and S_2. Since both P_0 and P_1 process their fraction during that time period, β^1 is separated into two fractions, α_0^1 and α_1^1, where α_j^i denotes the fraction of load distributed to P_j from β^i. We assume that β_i is from one contiguous region of load and later this assumption will be relaxed. Note that β_i is divided into $i + 1$ fractions because there are $i + 1$ processors computing their fraction between S_i and S_{i+1}. Also, each processor, P_j, receives the load from $m - j$ different β^i. For example, P_0 computes its fraction from S_0 to T_{finish} so it receives the fractions from all β^i.

In DLT literature, α_i usually means the size of the fraction of the load and its position in load is not considered. Since β_i and α_j^i are from different parts of total load, their position needs to be taken account as well as their size. For their size, we use $|\beta_i|$ and $|\alpha_j^i|$ and $l(\cdot)$ and $u(\cdot)$ for the boundary of the range of fractions. For notational consistency we also use $|\alpha_i|$ to denote the size of fraction of load distributed using the usual sequential distribution although α_i means the same quantity. For summary, our notation β_i and α_j^i consist of pairs of values, $(|\beta_i|, l(\beta_i))$ and $(|\alpha_j^i|, l(\alpha_j^i))$, respectively,

and $u(\cdot) = u(\cdot) + |\cdot|$.

The values of $\{|\alpha_j^i|\}$ and $|\beta^i|$ can be obtained from $|\alpha_i|$ which can be obtained from well-known solutions of the literature of divisible load theory and can be calculated using recursive equations presented in Section 3.2.

From $\{|\alpha_i|\}$, $\{|\alpha_j^i|\}$ is calculated with the following equations

$$|\alpha_j| = \sum_{i=j}^{m} |\alpha_j^i|, \quad j = 0, \ldots, m \tag{3.23}$$

with a constraint,

$$|\alpha_0^i| w_0 T_{cp} = \cdots = |\alpha_j^i| w_i T_{cp},\ i = 0, \ldots, m,\ j = 1, \ldots, m \tag{3.24}$$

and,

$$|\beta^i| = \sum_{j=0}^{i} |\alpha_j^i|,\ i = 0, \ldots, m. \tag{3.25}$$

From (3.23),

$$\begin{pmatrix} |\alpha_0| \\ |\alpha_1| \\ \vdots \\ |\alpha_m| \end{pmatrix} = \begin{pmatrix} |\alpha_0^0| & |\alpha_0^1| & \dots & |\alpha_0^{m-1}| & |\alpha_0^m| \\ & |\alpha_1^1| & \dots & |\alpha_1^{m-1}| & |\alpha_1^m| \\ & & \vdots & \vdots & \vdots \\ & & & |\alpha_{m-1}^{m-1}| & |\alpha_{m-1}^m| \\ & & & & |\alpha_m^m| \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} |\alpha_0^0| & |\alpha_0^1| & \dots & |\alpha_0^{m-1}| & |\alpha_0^m| \\ & \frac{w_0}{w_1}|\alpha_0^1| & \dots & \frac{w_0}{w_{m-1}}|\alpha_0^{m-1}| & \frac{w_0}{w_m}|\alpha_0^m| \\ & & \vdots & \vdots & \vdots \\ & & & \frac{w_0}{w_{m-1}}|\alpha_0^{m-1}| & \frac{w_0}{w_{m-1}}|\alpha_0^m| \\ & & & & \frac{w_0}{w_m}|\alpha_0^m| \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \tag{3.26}$$

$$= \begin{pmatrix} 1 & 1 & \dots & 1 & 1 \\ & \frac{w_0}{w_1} & \dots & \frac{w_0}{w_{m-1}} & \frac{w_0}{w_m} \\ & & \vdots & \vdots & \vdots \\ & & & \frac{w_0}{w_{m-1}} & \frac{w_0}{w_{m-1}} \\ & & & & \frac{w_0}{w_m} \end{pmatrix} \begin{pmatrix} |\alpha_0^0| \\ |\alpha_0^1| \\ \vdots \\ |\alpha_0^m| \end{pmatrix}$$

Here, the second equality is from (3.24).

Taking the inverse of the $(m+1)*(m+1)$ matrix,

$$\begin{pmatrix} |\alpha_0^0| \\ |\alpha_0^1| \\ \vdots \\ |\alpha_0^m| \end{pmatrix} = \begin{pmatrix} 1 & 1 & \dots & 1 & 1 \\ & \frac{w_0}{w_1} & \dots & \frac{w_0}{w_{m-1}} & \frac{w_0}{w_m} \\ & & \vdots & \vdots & \vdots \\ & & & \frac{w_0}{w_{m-1}} & \frac{w_0}{w_m} \\ & & & & \frac{w_0}{w_m} \end{pmatrix}^{-1} \begin{pmatrix} |\alpha_0| \\ |\alpha_1| \\ \vdots \\ |\alpha_m| \end{pmatrix} \tag{3.27}$$

Once $\{|\alpha_0^0|, |\alpha_0^1|, \dots, |\alpha_0^m|\}$ is found, other $\{|\alpha_j^i|\}, j \neq 0$ and $|\beta^i|$ follows from (3.24) and (3.25).

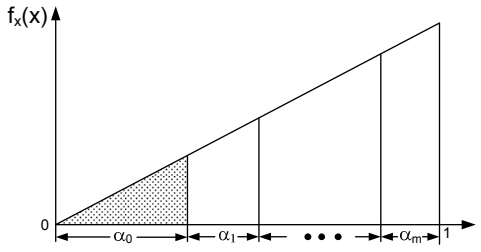
$$|\alpha_j^i| = |\alpha_0^i| \frac{w_0}{w_j} \tag{3.28}$$

The range for β^i is yet to be determined.

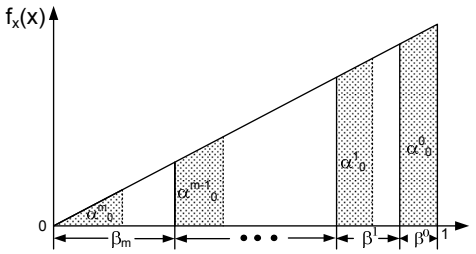
When the probability distribution function for the location of signatures is monotonically increasing or decreasing as shown in Fig. 3-9, for all β_i, one contiguous fraction suffices to give the optimal range. In the next subsection, we will give a motivational example with the probability distribution shown in the figure and discuss for the case of general shapes of distributions in the subsequent subsections.

3.4.2 Motivational Example

In this example, a linearly increasing probability density is considered. We describe a more efficient processing of the load based on the shape. Consider Fig. 3-9. It shows that the probability mass of the position of signatures linearly increases toward the end of the load. In the usual sequential load distribution, the load is partitioned



(a) Sequential distribution



(b) Distribution based on the probability mass

Figure 3-9: A comparison between the load distribution schemes

in sequence as shown in Fig. 3-9(a). The load distributed to α_0 is chosen from the beginning of the whole load of which computation begins at S_0. Fig. 3-9(b) is a distribution scheme based on the timing diagram of Fig. 3-8(b), where the fraction of the load containing the larger mass of probability of finding the signature is distributed to P_0 and processed between S_0 and S_1, during the earliest time. The grayed area of the figures identify the portion of load distributed to P_0. In the figure, the load from multiple ranges contribute to α_0, the load distributed to P_0. Note that, as probability mass monotonically increases, the range of β^i is continuous and each $\beta^i, i > 0$ is partitioned for multiple processors.

The probability distribution function of the position shown in the figure is given

as

$$f_x(x) = \frac{x}{2},\ 0 \leq x \leq 1 \tag{3.29}$$

m	0	1	2	3	4
$\lvert\alpha_m\rvert$	0.299	0.229	0.181	0.152	0.139
S_m	0	0.0701	0.117	0.146	0.160

Table 3.3: α_m and S_m for Linear Daisy Chain Network with $w = 1$, $z = 0.1$, $T_{cm} = T_{cp} = 1$

For a linear daisy chain network with 5 processors, the $\{|\alpha_m|\}$ and $\{S_m\}$ is calculated using recursive equations presented in section 3.2 and TABLE 3.2. The calculated values are shown in TABLE 3.3.

i	0	1	2	3	4
$\lvert\alpha_0^i\rvert$	0.0701154	0.0472423	0.0290935	0.0138541	0.138541
$\lvert\alpha_1^i\rvert$		0.0472423	0.0290935	0.0138541	0.138541
$\lvert\alpha_2^i\rvert$			0.0290935	0.0138541	0.138541
$\lvert\alpha_3^i\rvert$				0.0138541	0.138541
$\lvert\alpha_4^i\rvert$					0.138541
$\lvert\beta^i\rvert$	0.0701154	0.0944847	0.0872806	0.0554162	0.692703

Table 3.4: α_j^i, Linear Daisy Chain Network with $w = 1$, $z = 0.1$, $T_{cm} = T_{cp} = 1$

In TABLE 3.4, the size of the fraction of the load distributed to P_j from β^i, $\{|\alpha_j^i|\}$ are calculated using (3.27) and (3.28). Although the sizes are determined for α_j^i and β_i, with the aid of the DLT solution, the range of the fractions of load still needs to be determined. In Fig. 3-9(b), the shaded area indicates the fractions of load that need to be distributed to P_0. Here, the load distributed to P_0 are from multiple parts of the load based on their their probability mass. Now α_0^0 which is processed between S_0 and S_1 is taken from the part of load with highest probability mass. The fraction

of load processed between S_m and T_{finish}, β_m, is taken from the beginning part of the load because of their small probability mass. Since all the processors process the fraction the load during that period, β_m is partitioned and distributed to all the processors. As the range of fraction is dispersed in the load, the position of each fraction needs to be obtained. First, the range of β_i needs to be obtained and then the position of α_j^i is calculated.

$l(\beta^4)$	$l(\beta^3)$	$l(\beta^2)$	$l(\beta^1)$	$l(\beta^0)$
0	0.692703	0.748119	0.8354	0.929885

Table 3.5: Lower boundary of β^i with Linear Daisy Chain Network with $w = 1$, $z = 0.1$, $T_{cm} = T_{cp} = 1$, $u(\beta^i) = l(\beta^i) + |\beta^i|$

In this example with a monotonically increasing distribution, the range of β_i can be obtained in a straightforward manner with known $|\beta_i|$ as we can take β_i from the end of the load toward the beginning sequentially as shown in 3-9(b) and the calculated values are shown in TABLE 3.5. The table shows the lower boundary of β_i, but the upper boundary can be easily derived with $|\beta_i|$: $u(\beta_i) = l(\beta_i) + |\beta_i|$, where $u(\cdot)$ denote the upper boundary of the load fraction.

i	0	1	2	3	4
$l(\alpha_0^i)$	0.929885	0.8354	0.748119	0.692703	0
$l(\alpha_1^i)$		0.882642	0.777213	0.706557	0.138541
$l(\alpha_2^i)$			0.806306	0.720411	0.277082
$l(\alpha_3^i)$				0.734265	0.415623
$l(\alpha_4^i)$					0.554164

Table 3.6: Lower boundary of α_m^i, $l(\alpha_m^i)$ with Linear Daisy Chain Network with $w = 1$, $z = 0.1$, $T_{cm} = T_{cp} = 1$, $u(\alpha_m^i) = l(\alpha_m^i) + |\alpha_m^i|$

The lower boundary of $\{\alpha_m^i\}$ is shown in TABLE 3.6. Inside the range of β_i,

we partition the β_i in the order of the processor index for simplicity. Therefore, $l(\alpha_0^1) = l(\beta_1)$ and $l(\alpha_1^1) = l(\beta_1) + |\alpha_0^1|$ and other lower boundaries are obtained similarly. Other ways of partitioning β^i are possible but are not considered in this paper.

In order to find the expected signature search time for this distribution scheme, (3.5) can be re-written as

$$E[\mathbf{Y}] = \sum_{\{(m,i):\alpha_m^i \in \{\alpha_m^i\}\}} E[\mathbf{Y}|A_m^i] Pr(A_m^i) \tag{3.30}$$

where A_m^i denotes an event when the signature is found in α_m^i. Similarly to (3.7),

$$\begin{aligned} \mathbf{Y}|A_m^i &= g_m((X)|A_m^i) \\ &= (\mathbf{X}|A_m^i - l(\alpha_m^i))w_m T_{cp} + S_m \end{aligned} \tag{3.31}$$

where $l(\alpha_m^i)$ denotes the starting position of α_m^i in load.

Taking expectation of $\mathbf{Y}|A_m^i$ gives

$$E[\mathbf{Y}|A_m^i] = (E[\mathbf{X}|A_m^i] - l(\alpha_m^i))w_m T_{cp} + S_m. \tag{3.32}$$

The conditional expectation of the location of the signature given that it is distributed to P_m and from β^i,

$$E[\mathbf{X}|A_m^i] = \frac{\int_{l(\alpha_m^i)}^{u(\alpha_m^i)} x f_x(\zeta) d\zeta}{\int_{l(\alpha_m^i)}^{u(\alpha_m^i)} f_x(\zeta) d\zeta}. \tag{3.33}$$

Also, the probability of the event A_m^i is,

$$\Pr(A_m^i) = \int_{l(\alpha_m^i)}^{u(\alpha_m^i)} f_x(\zeta)d\zeta. \tag{3.34}$$

Finally, substituting the above equations into (3.30),

$$\begin{aligned} E[\mathbf{Y}] = \sum_{(m,i)} \left(\left(\frac{\int_{l(\alpha_m^i)}^{u(\alpha_m^i)} x f_x(\zeta)d\zeta}{\int_{l(\alpha_m^i)}^{u(\alpha_m^i)} f_x(\zeta)d\zeta} - l(\alpha_m^i) \right) w_m T_{cp} + S_m \right) \\ * \int_{l(\alpha_m^i)}^{u(\alpha_m^i)} f_x(\zeta)d\zeta \end{aligned} \tag{3.35}$$

When probability distribution is given as (3.29), we have,

$$\begin{aligned} \int_{l(\alpha_m^i)}^{u(\alpha_m^i)} x f_x(\zeta)d\zeta &= \int_{l(\alpha_m^i)}^{l(\alpha_m^i)+|\alpha_m^i|} \frac{x^2}{2}dx \\ &= \frac{3|\alpha_m^i|^2 l(\alpha_m^i) + 3|\alpha_m^i| l^2(\alpha_m^i) + |\alpha_m^i|^3}{6} \end{aligned} \tag{3.36}$$

and

$$\begin{aligned} \int_{l(\alpha_m^i)}^{u(\alpha_m^i)} f_x(x)dx &= \int_{l(\alpha_m^i)}^{l(\alpha_m^i)+|\alpha_m^i|} \frac{x}{2}dx \\ &= \frac{2|\alpha_m^i| l(\alpha_m^i) + |\alpha_m^i|^2}{4}. \end{aligned} \tag{3.37}$$

By using the calculated values from the above tables, we plot the expected search time using usual the sequential distribution in the DLT literature and when the load is classified by probability mass in Fig. 3-10. As shown in the figure, when the load is classified by the probability mass and scheduled with consideration of the starting time of each processor the expected time to find signatures is significantly faster. The

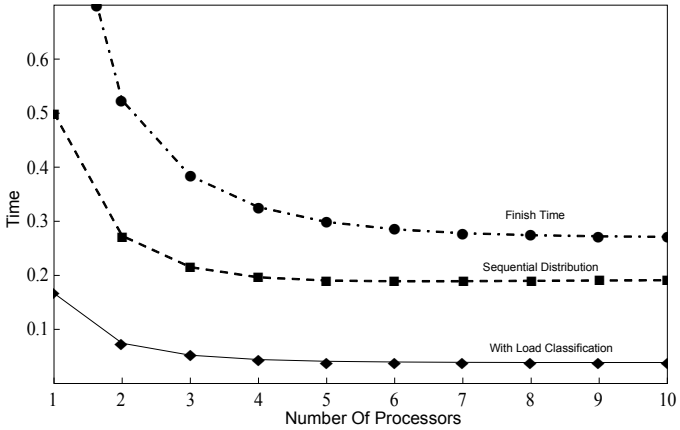


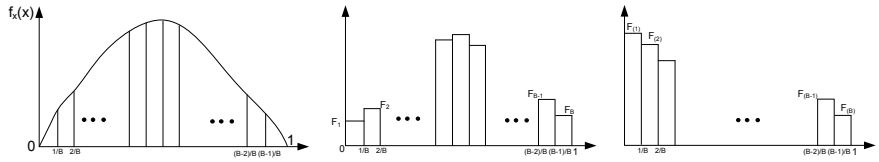
Figure 3-10: Comparison of expected time to find a single signature between sequential distribution and distribution based on load classification based on the probability distribution on single level tree network, $w_i = 1, z_i = 0.2, T_{cp} = 1, T_{cm} = 1$

finish time of computation is shown in the plot for comparison.

3.5 Load Rearrangement for Speeding-up Signature Search Time

3.5.1 Load Rearrangement Procedure

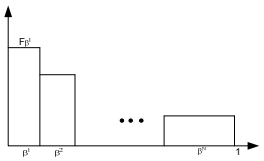
In this section, we present a streamlined procedure to distribute load based on the distribution of the signatures as introduced in the previous section. We assume that the probability distribution of the locations of the signatures is known. In this approach, the load is rearranged first in order of the likelihood of finding the signature. Fig. 3-11 describes the procedure. As shown in Fig. 3-11(a), the load is sliced into equal size *bin*s. We denote B the total number of bins and, therefore, the size of each bin is $1/B$. In the next step, shown in 3-11(b), the probability mass of each bin is



(a) Sequential Load Distribution

(b) Load partitioned into B bins

(c) Bins arranged in the order of the contained probability mass



(d) Probability mass approximation for β_i

Figure 3-11: Rearrangement of Loads based on Ranking

calculated as,

$$F_n = \int_{b_{n-1}}^{b_n} f_x(x)dx, \tag{3.38}$$

where F_n denotes the probability mass of nth bin and $b_n = n * \frac{1}{B}$. Once the mass of each bin is calculated the load is sliced and rearranged in the order of the decreasing magnitude of the probability mass as shown in Fig. 3-11(c). We call this the sorted load.

	β^0	β^1	β^2	β^3	β^4
$\lvert\beta^i\rvert$	0.0701154	0.0944847	0.0872806	0.0554162	0.692703
$u(\beta^i)$	0.0701154	0.1646	0.251881	0.307297	1

Table 3.7: $|\beta^i|$ and Upper boundaries of β^i with Linear Daisy Chain Network with $w = 1$, $z = 0.1$, $T_{cm} = T_{cp} = 1$

Finally, the load is distributed according to pre-calculated $\{\beta^i\}$. Once the load is

sorted according to the probability mass in the previous step, the range and the size of $\{\beta_i\}$ does not vary for a given network model with the same system parameters independently from the probability distribution of signatures. For example, the size and the range of $\{\beta^i\}$ can be calculated for the linear daisy network with $w = 1, z = 0.1$ and $T_{cm} = T_{cp} = 1$ as shown in TABLE 3.7. The size of β_i are obtained using (3.23), (3.24) and (3.25). The range of β_i is taken from the beginning of the sorted load. Therefore, the lower boundaries shown in TABLE 3.7 are the values in the sorted load, not in the original load.

3.5.2 Approximation of Expected Signature Search Time

Fig. 3-11(d) shows the procedure for obtaining approximated value of expected signature search time. The approximated value of the expected time of signature search time is given as,

$$\hat{E}[X] = \sum_{i=1}^{n} \frac{S_{i-1} + S_i}{2} * \hat{F}^i \tag{3.39}$$

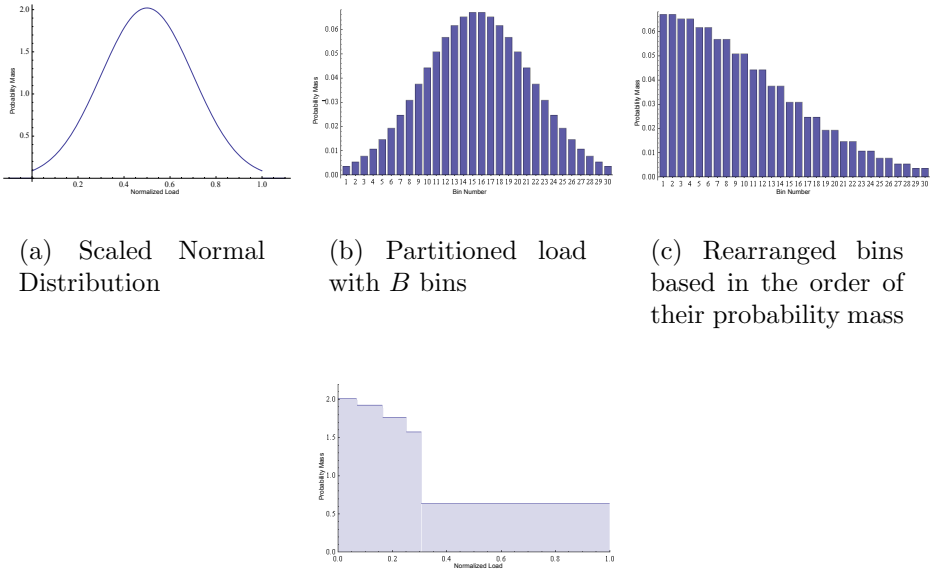
where $\hat{F}_i$ denotes the approximated probability mass calculated for $\{\beta^i\}$.

$$\hat{F}_i = F_{(k)}(b_k - l(\beta^i)) + \sum_{i=k}^{l} F_{(i)} + F_{(l)}(u(\beta^i) - b_l) \tag{3.40}$$

when $b_{k-1} \leq l(\beta^i) \leq b_k$ and $b_l \leq l(\beta^i) \leq b_{l+1}$.

Intuitively, this equation shows that the approximated expected signature search time is the weighted sum of the mid points of processing time of $\{\beta_i\}$ with their approximated probability masses.

3.6 Evaluation and Analysis



(a) Scaled Normal Distribution

(b) Partitioned load with B bins

(c) Rearranged bins based in the order of their probability mass

(d) Approximated Probability Mass $\hat{F}$

Figure 3-12: Rearrangement of Loads based on Ranking

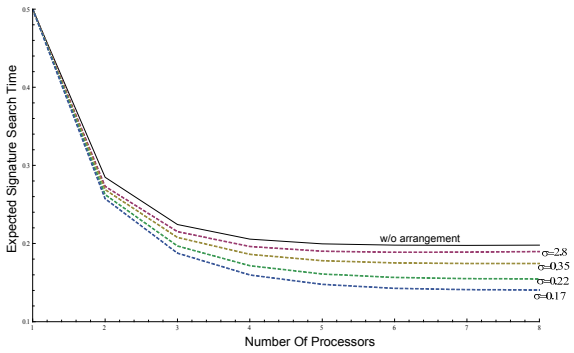
When the distribution of the location of signature time is given as a truncated normal distribution on the range $[0, 1]$ centered at the $1/2$, it can be written as,

$$f(x; \mu, \sigma, a, b) = \frac{\frac{1}{\sigma}\phi(\frac{x-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})}, \tag{3.41}$$

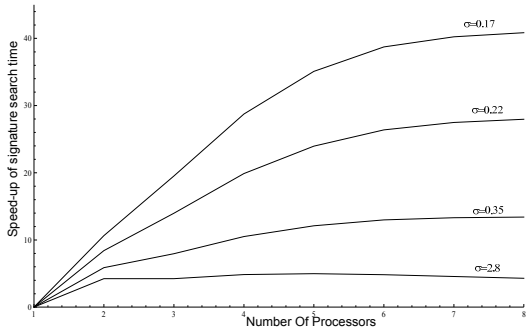
$\phi(\cdot)$ is the probability density function of the standard normal distribution, and $\Phi(\cdot)$ is its cumulative distribution function. Also, a and b denote the range of truncation, and μ denotes the average and σ is the standard deviation value. With $a = 0$, $b = 1$ and $\mu = 1/2$.

$$f(x; 1/2, \sigma, 0, 1) = \frac{\frac{1}{\sigma}\phi(\frac{x-1/2}{\sigma})}{\phi(\frac{1}{2\sigma}) - \phi(-\frac{1}{2\sigma})}, \tag{3.42}$$

This distribution is shown in Fig. 3-12(a). Following the procedure presented in the previous section, the load is sliced into bins as shown in Fig. 3-12(b) and rearranged based on the probability mass of the bins as shown in Fig. 3-12(c). The figures are based on when 30 bins being used. Finally, the approximated probability mass of β^i is plotted in Fig. 3-12(d).



(a) Expected signature search time



(b) Speed up of signature search time

Figure 3-13: Performance improvement(%) through load rearrangement for various σ

Fig. 3-13 shows the expected signature search time with and without load arrangement procedure with various standard deviation, σ. Fig. 3-13(b) shows the performance improvement in terms of percentage. As shown in the figures, as the probability of finding a signature is concentrated in a smaller area (lower σ value) the

performance gain increases.

3.7 Conclusion and Future Work

In the first part of this work, closed form solutions of the expected search time of the kth signature is derived. In the latter part, with a prior knowledge of the distribution of signatures, it is shown that expected time of finding a single signature can be improved and a procedure to speed up the signature search time and its simulation result is presented. With highly concentrated distributions, the improvement in the speed of finding a signature is shown to be significant. As an extension of this work, we will examine how database operations can be mapped to divisible load scheduling and how a prior knowledge of input data affects the performance and investigate how it can be used to make distribution strategies to improve performance.

The trend in data processing is the use of parallel processing to speed executing even to the level at a single machine (i.e. multicore architectures). Thus for radar and sensor data processing, the use of divisible load theory leads to a better quantitative understanding of processing performance and processing options. Thus the result in this paper should be of interest for quite some time.

Chapter 4

Analysis of Distributed Join Operation with Divisible Load Theory

4.1 Introduction

4.1.1 Distributed Join Algorithm

Join is an essential operation in relational database system as it usually precedes all of the operations relating two relations. Distributed join algorithm has been developed for a loosely-coupled distributed systems such as high speed local area networks. In the literature of database operations, two relations in a join operation are called R and S where the number of tuples in R is assumed to be smaller than S. Here, R and S are called the inner relation and the outer relation, respectively. In the following,

we introduce several versions of typical join operations on distributed environments.

Sort-Merge Join In Sort-Merge Join, the relations R and S are partitioned using a hash function h. The hash function takes the join attributes of a tuple as its input and give an index as its output. The hash function is applied to each tuple in the relation and the tuples with the same hash output is processed separately. First, the fraction of relations R and S with same hash output is sorted separately. The sorted relation is merged to find the matching join attribute to produce the join results. Optionally join condition can applied to filter out the join results.

Hash Join In join operation with hash table, the inner relation R is used to build a hash table using a hash function. For each tuple of the outer relation, S, its join attributes are hashed using a same hash function used for building the hash table. The hashed value of each tuple of S is used to look for tuples of R that where already loaded in the hash table - this process is called *hash probing*.

Grace hash and Hybrid hash join In both cases, the hash table is used for the join operation. However, before the join operation, both relations, R and S, are partitioned into N buckets using a hash function. Usually a bucket is partitioned into multiple disks for I/O efficiency.

4.1.2 Divisible Load Theory

Large sets of structured data such as database records can be modeled as divisible loads when the size of record is small enough compared to the total size of records under consideration. Divisible load theory was introduced by [29] and [30] and, also

independently in [31] as "large-grained parallelism". Ever since its introduction divisible load theory has served as effective modeling tools for data-intensive application [48] [49] [50] [7] [32] [3] and a large amount of work has been published for various network structures including linear networks [33], bus networks [34], single and multi level tree-networks [35] [36] [37], mesh networks [38] [39], hypercubes [40] and arbitrary graphs [41] with different constraints such as different release times, buffer constraints [42], communication start-up costs and time-varying network capacity. Also, various distribution strategies have been studied including multi-installment of load distribution [43]. In [44], [45] and [46], scheduling strategies without knowledge of network resources are examined.

4.1.3 Related Works

In [47] the modeling of the join operation as scheduling divisible tasks is verified by experiment in a cluster of workstations. In this work the relative error between the measurement from experiment and theoretical value from modeling decreases with sufficient size of data as transient irregularity of the computing and network environment such as the access time of network decreases. Other experimental work using divisible load theory has been studied. In [51] aligning biological sequences on distributed bus networks is presented. Divisible load analysis of computer vision and image processing was considered in [52] and [53]. In [54] handling of large-size discrete wavelet transform is studied in network-based computing systems. Large matrix-vector computation both in bus network is considered in [55] and [56].

4.1.4 System Model

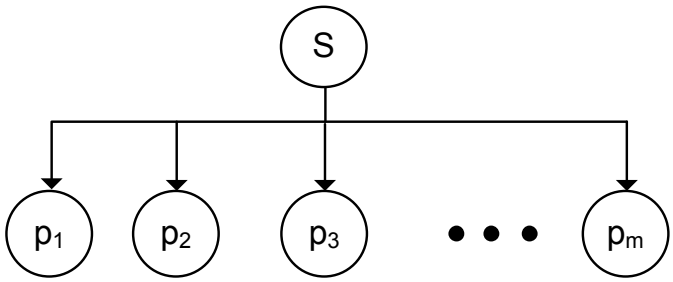


Figure 4-1: Model of Bus Network with Common Communication Links.

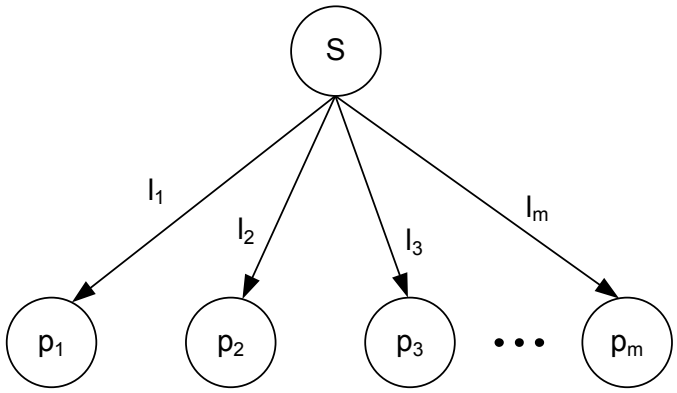


Figure 4-2: Model of Single Level Tree Network with Communication Links.

Fig. 4-1 and Fig. 4-2 describes a bus and a single level tree network model, respectively, used in our analysis. In the figures, there are m processors denoted by $p_1, p_1, \ldots, p_m$. In both network models, a scheduler, S, has access to a file system which stores relations and is responsible to calculate the size of fractions of relations to be distributed to each processor for distributed join operations. In the single level tree network model, S is connected to p_i with links l_i, $i = 1, \ldots, m$. In the bus network model, processors share a same bus channel. The inverse speed of each process is denoted by w_i and the inverse speed of each link is denoted by z_i. For the

bus network model, $z_i = z$ for all i.

p_i	Fraction of entire processing load assigned to ith processor
l_i	Fraction of entire processing load assigned to ith processor
w_i	Constant that is inversely proportional to computation speed of ith processor
z_i	Constant that is inversely proportional to communication speed of ith link
T_{cp}	Computation intensity: time taken to process a unit load on ith processor when $w_i = 1$.
T_{cm}	Communication intensity: time taken to communicate a unit load over link l_i when $z_i = 1$
α_i	Fraction of entire processing load assigned to ith processor
R,S	Input Relations of a join operation
$\|R\|$,$\|S\|$	Number of tuples in relations
ρ	Selectivity parameter of join operation
β	Ratio between the size of two relations $\|R\|/\|S\|$

Table 4.1: Summary of notation for Divisible Load Theory

TABLE 4.1 summarizes the notation used throughout this paper. It is assumed that the relations used in the operation are available on S, and each processor starts computing only when the transfer of the load is finished. Also, the scheduler engages in communication to only a single link at a time.

4.2 Divisible Load Analysis of Distributed Join Algorithm

4.2.1 Cost of Operations

A distributed operation consists of four phases. In the first phase, with a given size of a load, the size of the fractions of load to be distributed to each processor is decided by the scheduler in a *scheduling phase*. The next phase is is *communication phase* where the fractions of load are transfered to processors. Each processor applies the operation on its fraction in a *computation phase*. When the results of operation need to be reported back for further processing *report phase* follows. In join operations, the result is another relation which almost always needs to be presented to the user of the system or saved for further operations such as another join. Therefore, we consider the report phase explicitly in the following formulation. We denote two input relations as R and S which contain the tuples, the records consists of the values of multiple fields.

We assume that a selection operation is applied before distributing relations so that only the fields that appear in the result of the operation is distributed. We further assume that the aggregate size of fields from each relation is same so the size of the relations are only expressed in terms of number of tuples in the relations. Without loss of generality, we assume the size of R, denoted as $\|R\|$ is smaller than the size of S, $\|S\|$. The smaller relation R is called *inner relation* and S is called

outer relation. We denote the ratio of these two sizes as β.

$$\beta = \|R\|/\|S\|,\ 0 < \beta \leq 1 \tag{4.1}$$

For the report phase, we define a selectivity factor, ρ, which is defined as the ratio of the number of tuples that satisfy the join condition to the number of tuples in the Cartesian product of two relations.

We describe the cost function for each phase for p_i in terms of the number of tuples of two relations scheduled in the phase. We ignore the time in scheduling phase which is negligible compared to the actual communication and computation phases. Actual cost depends on the join types and distribution strategy as well as system parameters such as the communication channel speed and the computation speed of each processor.

1. Distribution cost, $D_i(\|R_i\|, \|S_i\|)$

2. Computation cost, $C_i(\|R_i\|, \|S_i\|)$

3. Report cost, $M_i(\|R_i\|, \|S_i\|)$

The cost function described above gives the number of tuples as its output and communication and computation time is linearly proportional to them.

1. Distribution communication time

$$D_i * z_i * T_{cm} \tag{4.2}$$

2. Computation time of ith processor, p_i

$$C_i * w_i * T_{cp} \tag{4.3}$$

3. Join result communication time

$$M_i * z_i * T_{cm} \tag{4.4}$$

Here, z_i denotes the inverse speed of communication speed of link from p_0 to p_i and w_i denotes the inverse speed of computation speed of p_i both in terms of time for a tuple.

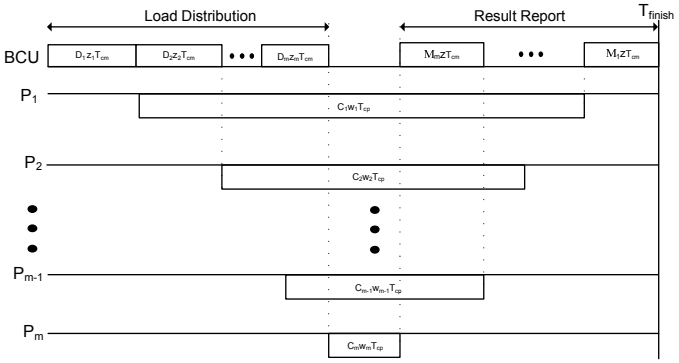


Figure 4-3: Timing Diagram of Distributed Join Operation

Fig. 4-3 shows the timing diagram including communication from processors to scheduler with joined tuples. Besides communication time that force the processors to be idle, all the processors are fully utilized. Using the diagram, the following recursive equations can be derived. With a single level tree network, the general form

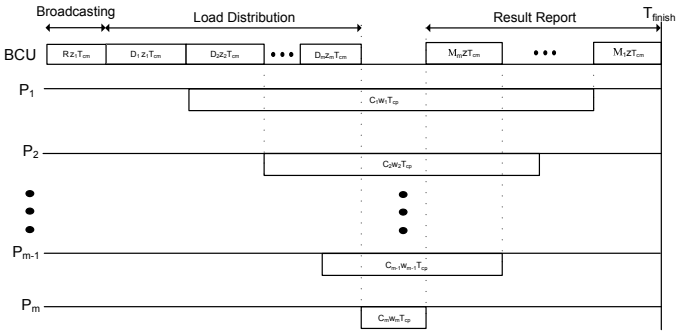


Figure 4-4: Timing Diagram of Distributed Join Operation with Broadcasting Support

of recursive equation is given as,

$$C_i w_i T_{cp} = D_{i+1} z_{i+1} T_{cm} + C_{i+1} w_{i+1} T_{cp} + M_i z_{i+1} T_{cm} \qquad i = 1, 2, \ldots, m-1 \tag{4.5}$$

Fig. 4-4 shows initial broadcasting phase when the inner relation is communicated to all the processors, which is only available for bus networks.

A bus network can be considered as a special form of a single level tree network where $z_i = z$ for l_i.

$$C_i w_i T_{cp} = D_{i+1} z T_{cm} + C_{i+1} w_{i+1} T_{cp} + M_i z T_{cm} \qquad i = 1, 2, \ldots, m-1 \tag{4.6}$$

Note that cost functions, C_i,D_i and M_i depends on the size of the fractions of two relations, $\|R_i\|$ and $\|S_i\|$, which is expressed as the fraction of the total number of tuples in the relations, $\alpha_i\|R\|$ or $\alpha_i\|S\|$. The optimal set of $\{\alpha_i\}$ can be obtained

using the recursive equations above along with the normalization equation,

$$\sum_{j=1}^{m} \alpha_i = 1 \tag{4.7}$$

From Fig. 4-3 the finish time of operation without broadcasting of the inner-relations is obtained as,

$$T_{finish} = D_1 z_1 T_{cm} + C_1 w_1 T_{cp} + M_1 z_1 T_{cm} \tag{4.8}$$

This is because the finish time of total operations for a single-level tree network is the same as when the first processor, P_1, finishes

In the following subsections, specific join algorithms will be examined and the optimal distribution of relations are obtained using divisible load theory.

4.2.2 Distributed Inner Loop Join

When the inner loop is used for the join operation, the smaller relation, R, has to be shipped to each processor while the bigger relation can be partitioned into m fractions for parallel processing. With a bus network, the inner relation can be broadcast to each processor before the joining operation on each processor if broadcasting is supported by the communication medium. In an inner loop styled join, each tuple in one relation is compared with all the tuples of another relation. Therefore the cost of operation on ith processor is given as,

$$C_i = \|R\| * \|S_i\| = \beta \alpha_i \|S\|^2 \tag{4.9}$$

For cost of reporting back the result of join operation is proportional to the multiplication of the size of two relations on each processor and to the selectivity factor, ρ. Therefore,

$$M_i = \rho\|R\| * \|S_i\| = \rho\beta\|S\| * \alpha_i\|S\| = \rho\beta\alpha_i\|S\|^2 \tag{4.10}$$

The cost of distribution is given as,

$$D_i = \|R\| + \|S_i\| = \beta\|S\| + \alpha_i\|S\| = (\beta + \alpha_i)\|S\| \tag{4.11}$$

When broadcasting of inner relation is supported,

$$D_i = \|S_i\| = \alpha_i\|S\| \tag{4.12}$$

with time to broadcasting the inner relation R before the parallel joining operations added to the total timespan.

Single Level Tree Network

With a single level tree network, the inner relation R as a whole is distributed to each processor sequentially along with the fraction of the outer relation S. From the timing diagram in Fig. 4-3 and using (4.5),(4.9),(4.10) and (4.11), the following

recursive equation can be obtained:

$$\begin{aligned}\alpha_i\beta\|S\|^2 w_i T_{cp} = (\alpha_{i+1}+\beta)\|S\| z_{i+1} T_{cm} \\ + \rho\alpha_{i+1}\beta\|S\|^2 z_{i+1} T_{cm} + \alpha_{i+1}\beta\|S\|^2 w_{i+1} T_{cp}, \\ i = 1, \ldots, m-1,\end{aligned}$$

which can be re-written to

$$\begin{aligned}\alpha_i\beta w_i T_{cp} = \alpha_{i+1}(\frac{z_{i+1}T_{cm}}{\|S\|} + \rho\beta z_{i+1}T_{cm} + \beta w_{i+1}T_{cp}) \\ + \frac{\beta z_{i+1}T_{cm}}{\|S\|},\ i = 1, 2, \ldots, m-1\end{aligned} \tag{4.13}$$

or

$$\alpha_i = \alpha_{i+1} f_{i+1} + \xi_{i+1},\ i = 1, \ldots, m-1 \tag{4.14}$$

where

$$\begin{aligned}f_{i+1} &= (\frac{z_{i+1}T_{cm}}{\|S\|} + \rho\beta z_{i+1}T_{cm} + \beta w_{i+1}T_{cp})/\beta w_i T_{cp} \\ &= \frac{w_{i+1} + z_{i+1}\delta}{w_i},\ i = 1, 2, \ldots, m-1\end{aligned} \tag{4.15}$$

with

$$\delta = \left(\frac{1}{\|S\|\beta} + \rho\right)\frac{T_{cm}}{T_{cp}} \tag{4.16}$$

and

$$\xi_{i+1} = \frac{z_{i+1}T_{cm}}{\|S\| w_i T_{cp}},\ i = 1, 2, \ldots, m-1. \tag{4.17}$$

Each fraction can be expressed in terms of α_m,

$$\alpha_i = \alpha_m K_i + L_i,\ i = 1, \ldots, m-1 \tag{4.18}$$

where

$$K_i = \prod_{j=i+1}^{m} f_j, \quad i = 1, \ldots, m-1 \tag{4.19}$$

and

$$L_i = \sum_{k=i+1}^{m} \xi_k \left(\prod_{j=i+1}^{k-1} f_j \right), \quad i = 1, \ldots, m-1 \tag{4.20}$$

Note that above derivation in terms of α_m follows the analysis presented in [53], where communication start-up cost is considered. In our case, communication cost to transport inner relation R can be considered as a fixed start up cost in equations. With the normalization equation, (4.7), we have,

$$\alpha_m = \left(\frac{1 - \sum_{i=1}^{m-1} L_i}{1 + \sum_{i=1}^{m-1} K_i} \right) \tag{4.21}$$

or

$$\alpha_m = \left(\frac{1 - \eta(m)}{\Omega(m)} \right) \tag{4.22}$$

where, following the notation in [53],

$$\eta(m) = \sum_{i=1}^{m-1} \sum_{k=i+1}^{m} \xi_k \left(\prod_{j=i+1}^{k-1} f_j \right) \tag{4.23}$$

and

$$\Omega(m) = \left(1 + \sum_{i=1}^{m-1} \prod_{j=i+1}^{m} f_j\right), \tag{4.24}$$

In [53], $\eta(m)$, is referred as *overhead factor* because it gives a necessary and sufficient condition for the existence of an optimal load distribution using all the m processors when m processors are available for parallel processing.

The condition is given as,

$$\eta(m) < 1 \tag{4.25}$$

which is derived from $\alpha_m > 0$.

To be precise, since the number of tuples allocated to a processor cannot be less than 1. From

$$\|R\|\alpha_m = \|R\| \left(\frac{1 - \eta(m)}{\Omega(m)}\right) \geq 1 \tag{4.26}$$

$$\eta(m) \leq 1 - \frac{\Omega(m)}{\|R\|} \tag{4.27}$$

From (4.8), (4.9), (4.10) and 4.11 the finish time of the operation can be obtained as,

$$T_{finish} = \beta\alpha_1|S|^2 w_1 T_{cp} + (\beta + \alpha_1)\|S\| z_1 T_{cm} + \rho\beta\alpha_1\|S\|^2 z_1 T_{cm} \tag{4.28}$$

Since $\eta(m)$ increases as m increase as shown in Fig. 4-5, there is an m^* satisfying (4.25) where $m^* \leq m$ for a fixed sequence. Adding more processors beyond m^* would not decrease the finish time as the optimal solution does not exist.

Once m^* is found, for a fixed sequence of distribution, $\{\alpha_1, \alpha_2, \ldots, \alpha_{m^*}\}$, gives the optimal solution as it fully utilizes all the processors and links as shown in Fig.

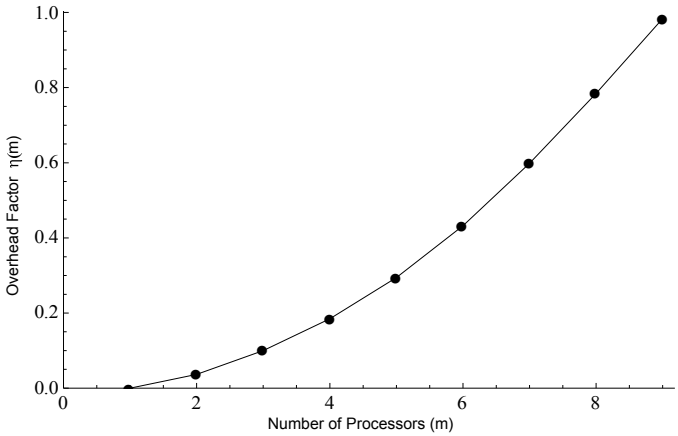


Figure 4-5: Overhead Factor $\eta(m)$

(4-3). If all the sequences are considered, finding the subset of processors giving optimal solution is conjectured to be an NP-hard problem in [57], where a fixed cost of communication start-up cost was considered. In our case, the time to transport $|R|$ to each processor ($\beta|S|z_{i+1}T_{cm}$) can be considered as a start up cost for communication although it varies with the communication speed, z_i. Finding an optimal sequence amounts to searching $m!$ space which is not practically tractable. In [53], a greedy algorithm is suggested, which makes use of a maximal set of processors in the order of their decreasing computation speeds, *Fast Sequence*, while satisfying the overhead factor condition, (4.23). In the following analysis, we adopt this greedy algorithm as it gives a good suboptimal solution. For more detailed discussion, the reader is referred to [53] and [7].

When the whole load is processed by only one processor, with $\alpha_1 = 1$ in (4.28),

$$T^1_{finish} = \beta|S|^2 w_1 T_{cp} + (\beta+1)\|S\| z_1 T_{cm} + \rho\beta\|S\|^2 z_1 T_{cm} \tag{4.29}$$

Speed-up, Γ is the value to quantify the advantage of parallel processing and defined as the ratio between finish time with parallel processing to the finish time when the whole load is processed by only one processor.

$$\Gamma = \frac{T^1_{finish}}{T_{finish}} \tag{4.30}$$

The computation speed of the first processor is used under assumption that the load distribution is in the order of decreasing speed, so p_1 is the fastest.

Bus Network with broadcasting support

When a bus network is used, we can have same the solution as (4.21) with $z_i = z$ for all i. When broadcasting is supported, the inner relation R, can be communicated to all of the processors concurrently. After the inner relation is broadcast, the fraction of outer relation is distributed to processors sequentially.

The recursive equation for the part of distributing the outer relation is given as,

$$\begin{aligned}\alpha_i\beta|S|^2 w_i T_{cp} = \alpha_{i+1}|S|z_{i+1}T_{cm} + \rho\alpha_{i+1}\beta|S|^2 z_{i+1}T_{cm} + \\ \alpha_{i+1}\beta|S|^2 w_{i+1}T_{cp},\end{aligned} \tag{4.31}$$

which can be reduced to

$$\alpha_i = \alpha_{i+1} f_{i+1}, \tag{4.32}$$

where f_{i+1} is given in (4.16).

The closed form solution can be obtained as,

$$\alpha_i = \alpha_m K_i \tag{4.33}$$

where K_i is given in (4.19).

The closed form of the fraction for the last processor is given as,

$$\alpha_m = \frac{1}{\Omega(m)} \tag{4.34}$$

where $\Omega(m)$ is from (4.24). The last fraction is always greater than 0. However, it should be true that $\|S\| * \alpha_m > 1$, since the number of tuples less than 1 is meaningless. Therefore, for a fixed sequence, m should be chosen to satisfy,

$$\|S\| > \Omega(m) \tag{4.35}$$

The finish time includes the time to broadcast the inner relations, R.

$$T_{finish} = \|R\| z T_{cm} + \alpha_1 \beta \|S\|^2 w_1 T_{cp} + \alpha_1 \rho \beta \|S\|^2 z_1 T_{cm} \tag{4.36}$$

The speed-up is given as,

$$\Gamma = \frac{\|R\| z T_{cm} + \beta \|S\|^2 w_1 T_{cp} + \rho \beta \|S\|^2 z_1 T_{cm}}{\|R\| z T_{cm} + \alpha_1 \beta \|S\|^2 w_1 T_{cp} + \alpha_1 \rho \beta \|S\|^2 z_1 T_{cm}} \tag{4.37}$$

$\|R\| = 10,000, \qquad \|S\| = 100,000$
$m = 25, \quad z_i = 10^{-5}, \quad T_{cp} = T_{cm} = 1$
$\beta = 10, \quad \rho = 2 * 10^{-6}$
$w_i (i = 1, \ldots, 25) =$
$\{0.272, 0.546, 0.802, 0.816, 0.823,$
$0.935, 1.52, 2.43, 2.58, 2.77, 2.99,$
$3.24, 3.36, 3.73, 4.9, 4.98, 5.49,$
$5.61, 6.48, 8.11, 8.43, 9.27, 9.66, 9.7, 9.72\}(10^{-9})$

Table 4.2: Summary of parameters for evaluation of distributed inner-loop join

Evaluation

The evaluation is based on the standard Wisconsin Benchmark [58]. In *joinABprime*, a 1,000,000 tuple relation is joined with a 100,000 tuple relation and produces a 100,000 tuple result relation, T. $\beta = \|S\|/\|R\| = 1000000/100000 = 10$ and ρ is calculated as $\rho = \|T\|/(\|R\| * \|S\|) = 100000/(1000000 * 100000) = 2 * 10^{-6}$. We choose $T_{cm} = T_{cp} = 1$ and $z = 10^{-5}$ for all p_i. TABLE 4.2 summarizes the parameters used in our evaluation.

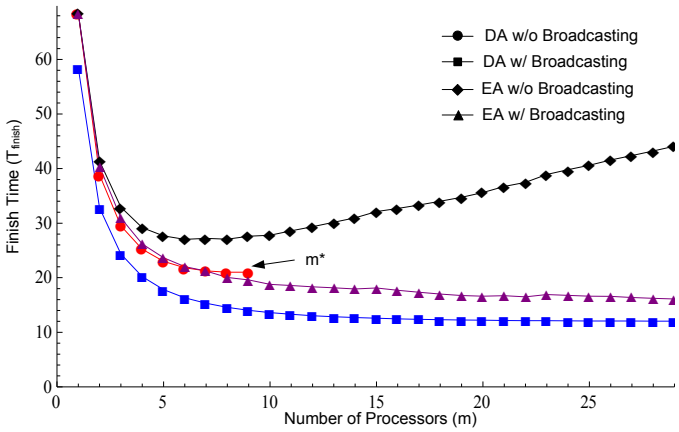


Figure 4-6: The Comparison of Finish Time against the Equal Allocation Scheme

Fig. 4-6 and Fig.4-7 compare the finish time and the speed up of four cases. In the figure, DA stands for the allocation based on divisible load analysis presented in

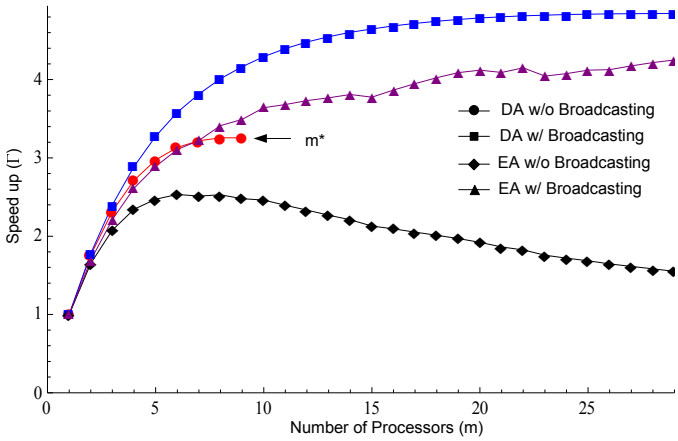


Figure 4-7: Speed Up

the previous section while EA stands for the equal allocation scheme. For each allocation scheme, the availability of the support of broadcasting of inner relationship is considered. As shown in the figure, when broadcasting is supported, the performance is better in both DA and EA. For DA with broadcasting, the maximum number of processors that can be utilized is limited and shown as m^*. For the equal allocation scheme, the allocation is in the order of decreasing speed of the processors. For the report phase, the processor that finishes the computation early has a priority to send back the result to the BCU. In [59], the analysis of equal allocation scheme is presented. There also exists the maximum number of processors that can be utilized with the equal allocation without broadcasting support.

4.2.3 Distributed Sort-Merge Join

In distributed sort merge join operation, the same hash function is applied to both the inner and the outer relations to find the processor to where the relations is distributed. Fig. 4-8 describes the process of partitioning the inner relation, R, into m fractions.

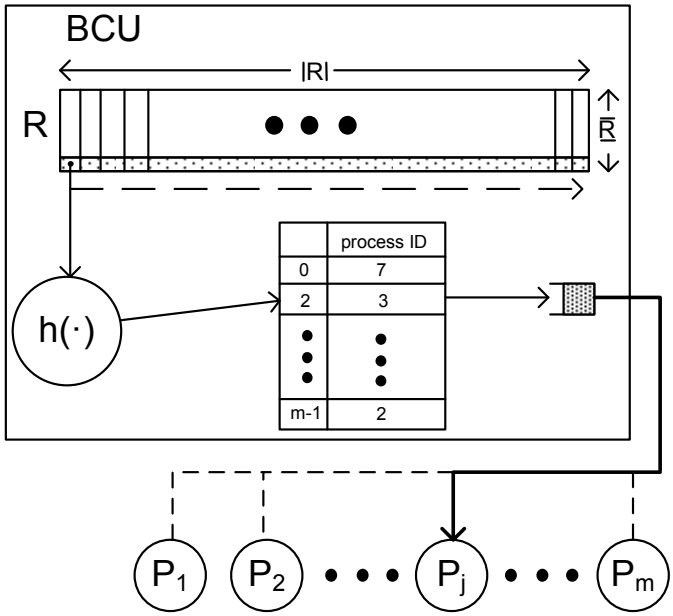


Figure 4-8: Distributed Sort Merge Process

The same process applies to the outer relation, S, as well. A hash function is chosen to take the join attributes of the relation and gives an integer value in $[1, m]$. This index is used to find the process id. The tuples with attributes having the same hashed values are accumulated in local buffers. When all the tuples are processed, they are distributed to the corresponding processor in one message. Since each processor receives the tuples with the same hash attributes, the result of join operations can be fully obtained at each processor and can be combined without any intermediate communication between the processors. When the values of join attributes exhibits uniformity, which is good assumption as described in [27], the number of tuples to be distributed to each processor will be approximately the same as a good hash function gives uniformly distributed hash values. However, when the communication cost is not negligible such as in local network environments, equal load distribution does

not give an optimal time span [7]. For the case where the capacity of processor and communication link is not known for optimal allocation, in [59] the equal allocation scheme of divisible load is studied. In this paper, we assume that we have the full knowledge of computation speed of each processor and communication speed of each link, so the optimal allocation can be calculated. Once the optimal allocation is calculated, the hash function should be designed to produce an optimal allocation.

Once the fraction of relations is distributed to each processor, the local operation involves the sorting of two fractions of relations and merging operation which is a linear scan of two partial relations. In the merge operations, the matching tuples from each relation become the output of the join operations. We assume that both relations are already in sorted order. It is a plausible assumption as sorting needs to be required only once and other typical operation such as inserting, deleting etc. for the sorted relation can gain significant performance gain. Therefore, in the following analysis the cost for sorting tuples will not be considered.

Assuming that the join attributes in both relations R and S exhibit a uniform distribution, the size of tuples allocated to each processor will be the same size. In the following derivation, we will derive the optimal fraction of relations based on the DLT optimal criterion and will show a procedure to modify the hash process to accommodate the optimal allocation instead of equal allocation. We have

$$\|R_i\| = \alpha_i \|R\|, \quad \|S_i\| = \alpha_i \|S\| \tag{4.38}$$

The computation cost on each processor consists of sorting and linear merging

phases. Since sorting of tuples needs to be done only once and is not required in subsequent process, we only consider the computation cost linear merge operation:

$$C_i^{sm}(R_i, S_i) = \|R_i\| + \|S_i\| = \alpha_i\|R\| + \alpha_i\|S\| = \alpha_i(\beta+1)\|S\| \tag{4.39}$$

The distribution and report cost are given as,

$$\begin{aligned} D_i^{sm} &= \alpha_i(\beta+1)\|S\| \\ M_i^{sm} &= \rho\|R_i\| * \|S_i\| = \rho\beta\alpha_i^2\|S\|^2 \end{aligned} \tag{4.40}$$

Using (4.5) with the above cost functions, the following recursive equation can be obtained,

$$\begin{aligned} \alpha_i\|S\|(\beta+1)w_iT_{cp} = \alpha_{i+1}\|S\|(\beta+1)z_{i+1}T_{cm} + \\ \alpha_{i+1}\|S\|(\beta+1)w_{i+1}T_{cp} + \rho\beta\alpha_{i+1}^2\|S\|^2 z_{i+1}T_{cm} + \\ i = 1, 2, \ldots, m-1, \end{aligned} \tag{4.41}$$

which can be rewritten to,

$$\begin{aligned} \alpha_i &= \alpha_{i+1}\frac{z_{i+1}T_{cm}}{w_iT_{cp}} + \alpha_{i+1}\frac{w_{i+1}}{w_i} + \\ &\qquad \alpha_{i+1}^2\frac{\rho\beta\|S\|z_{i+1}T_{cm}}{(\beta+1)w_iT_{cp}} \\ &= \alpha_{i+1}\xi_{i+1} + \alpha_{i+1}^2\frac{\rho\beta\|S\|}{\beta+1}\sigma_{i+1} \\ &\qquad i = 1, 2, \ldots, m-1, \end{aligned} \tag{4.42}$$

where $\xi_{i+1} = \sigma_{i+1} + \eta_{i+1}$ and

$$\sigma_{i+1} = \frac{z_{i+1} T_{cm}}{w_i T_{cp}} \quad \text{and} \quad \eta_{i+1} = \frac{w_{i+1}}{w_i} \qquad i = 1, 2, \ldots, m-1. \tag{4.43}$$

With the above recursive equation and the normalization equation given in (4.7), we can setup the following matrix equation,

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \vdots \\ \alpha_{m-1} \\ 1 \end{bmatrix} = \frac{\rho\beta\|S\|}{\beta+1} \begin{bmatrix} 0 & \sigma_2 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & \ldots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 0 & \sigma_m \\ 0 & 0 & 0 & 0 & \ldots & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1^2 \\ \alpha_2^2 \\ \alpha_3^2 \\ \alpha_4^2 \\ \vdots \\ \alpha_{m-1}^2 \\ \alpha_m^2 \end{bmatrix} + \begin{bmatrix} 0 & \xi_2 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \xi_3 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & \xi_4 & \ldots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 0 & \xi_m \\ 1 & 1 & 1 & 1 & \ldots & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \vdots \\ \alpha_{m-1} \\ \alpha_m \end{bmatrix} \tag{4.44}$$

This simultaneous non-linear eqaution can be solved using standard iterative numerical methods. One of them is *Newton-Raphson Method* used in our analysis [60].

The above simultaneous equation can be rewritten to in an implicit form:

$$\mathbf{F} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \vdots \\ \alpha_{m-1} \\ 1 \end{bmatrix} - K \begin{bmatrix} 0 & \sigma_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & \sigma_m \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1^2 \\ \alpha_2^2 \\ \alpha_3^2 \\ \alpha_4^2 \\ \vdots \\ \alpha_{m-1}^2 \\ \alpha_m^2 \end{bmatrix} - \begin{bmatrix} 0 & \xi_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \xi_3 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \xi_4 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & \xi_m \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \vdots \\ \alpha_{m-1} \\ \alpha_m \end{bmatrix} = 0, \tag{4.45}$$

or

$$F_i = \begin{cases} \alpha_i - K\sigma_{i+1}\alpha_{i+1}^2 - \xi_{i+1}\alpha_{i+1} & \text{i = 1,...,m-1} \\ 1 - (\alpha_1 + \cdots + \alpha_m) & \text{i = m} \end{cases} \tag{4.46}$$

where $K = \frac{\rho\beta\|S\|}{\beta+1}$.

The *Jacobian* matrix is given as

$$J_{ij} = \begin{cases} 1 & \text{i = j} \\ -\xi_j - 2\alpha_j K\sigma_j & \text{i = j-1} \\ -1 & \text{i = m} \end{cases} \tag{4.47}$$

using

$$J_{ij} = \frac{\partial F_i}{\partial \alpha_j} \tag{4.48}$$

With an initial estimation vector $\bar{\alpha}$, new estimation can be obtained using,

$$\bar{\alpha}_{new} = \bar{\alpha}_{old} + \delta\bar{\alpha} \tag{4.49}$$

where the correction vector can be obtained as,

$$\delta\bar{\alpha} = \mathbf{J}^{-1} \cdot -\mathbf{F} \tag{4.50}$$

The last row of the matrix equation incorporates the normalization equation and the other rows are the matrix form of (4.42). The solution can be found standard iterative method by inserting initial estimation of $\{\alpha_i\}$ to the right hand side and obtain the new vector in the left hand side. This process is known to iterate to convergence.

From (4.42),

$$\frac{d\alpha_i}{d\alpha_{i+1}} = \xi_{i+1} + 2\alpha_{i+1}C\sigma_{i+1} > 0 \tag{4.51}$$

when $\alpha_{i+1} >= 0$, where $C = \frac{\rho\beta\|S\|}{\beta+1} > 0$. Therefore $\{\alpha_i\}$ monotonically decreases. The iterative procedure can stop when $\alpha_m * \beta * |R| < 1$ or the difference between the result of current step and the previous step is less than a specified constant, whichever comes first. In (4.51), C is a value greater than 0.

Finish time and speed up is given as,

$$T_{finish} = \alpha_1(\beta + 1)\|S\|w_1T_{cp}, \tag{4.52}$$

and

$$\Gamma = \frac{1}{\alpha_1}, \tag{4.53}$$

respectively.

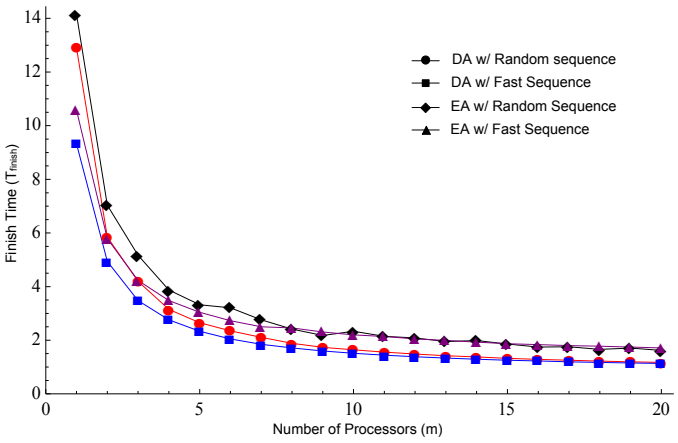


Figure 4-9: The comparison of the Finish Time of the Distributed Sort Merge Join operation

Fig. 4-9 and Fig. 4-10 shows the finish time and the speed-up of distributed sort merge process. The inverse speed of the processors are chosen randomly from the range of $0.8 * 10^{-5}\tilde{1}.2 * 10^{-5}$ and the inverse speed of communication z is chosen from

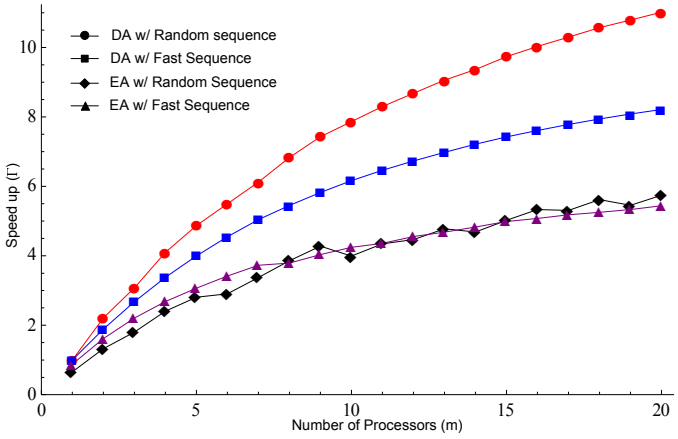


Figure 4-10: Speed up of distributed sort merge

the range of $0.8 * 10^{-6}\tilde{1}.2 * 10^{-6}$. Other parameters follow TABLE 4.2. The figures show the allocation based on the divisible analysis and equall allocation. For each allocation scheme, the fast sequence is compared the against the random sequence.

The sequence of operation following the order of the decreasing speed of processors shows the persistent improvement over when the sequencing is arbitrary. The performance of the allocation based on divisible load analysis is shown to be better than that of equal allocation. For the computation of finish time of the equal allocation scheme, we use the first come first served policy for the report phase.

4.3 Conclusion

We applied divisible load theory to the area of distributed database systems. We consider the distributed join operation for various distributed algorithm and showed that by allocating tuples with consideration of communication cost will give an optimal solution in terms of minimal operation time. We compared our result with an equal

allocation scheme. Database technology is an essential component of modern computer systems. The type of analysis presented in the paper allows both performance prediction and the development of efficient database algorithms. The work discussed here is very fundamental so the results should be of interest for some time.

Chapter 5

Multiple Mobile Robot Dispatch Strategy for Cooperative Applications

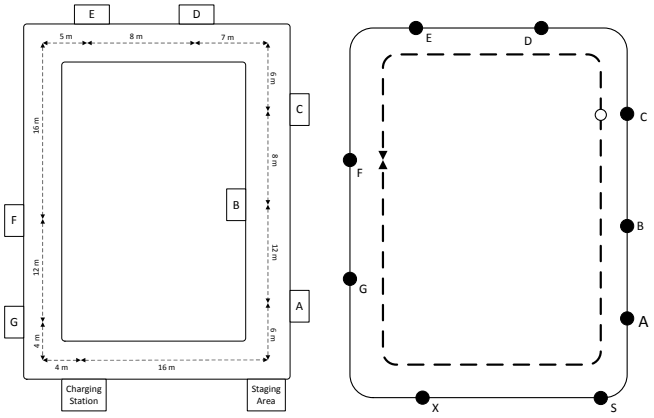
5.1 Introduction

Delivery applications for multiple mobile robots find great importance in hospital or factory environments where specimen or parts need to be delivered by interactive requests. In such environments, timely delivery of items is the most important requirement while avoiding unnecessary robot movement to minimize the impact of battery charging time and increase overall life span of the mobile robots. In such environments, the robots follow well-defined paths such as corridors in a building or a guided line on the factory floor which allows a path based model of robot movement that allows the effective computation of planning of robot movement. In this work we

propose path based mobile robot dispatch strategy for delivery request between two static locations. In the next section, an application model and problem description is described. In the following section, path based cost analysis that is used for making dispatch decision is introduced. In the section 5.3.4 dispatch strategy is proposed and it is evaluated in the section 5.4. Lastly, in the section 5.5 the conclusion is made with future directions of our work.

5.2 Application Model and Problem Description

5.2.1 Application Model



(a) Floor Map (b) Path Model

Figure 5-1: Floor map of multiple offices and its path view. Locations or offices are modeled as nodes in the path view. Two paths form the office C to F are shown in the path view.

We consider a delivery application with multiple robots delivering items between offices in a building environment. In Fig. 5-1 an actual floor plan and its path model of the map are shown. In path model for each pair of locations a unique path is

determined with the distance between them. A path is defined with a sequence of locations where the distance of between each location is given.

Three different kinds of locations are shown. First, task locations are where items are picked up from of drooped off for delivery, $A \sim G$, in the figure. Second, charging station is a place where a robot charges its battery, indicated as X in the figure. Third, staging area, S, where the robots are staged in the beginning of the system.

A delivery task request dynamically arrives to the dispatch system and each of which is defined with a unique pick-up and drop-off location. It is assumed that the pick-up and the drop-off locations are known to the dispatch system when the request is made.

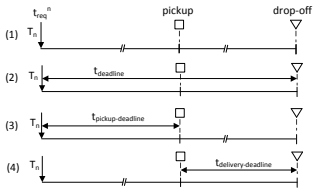


Figure 5-2: Illustration of three types of deadlines

For each delivery request, we consider three types of deadline for the delivery as shown in Fig. 5-2. In the first case, (2) in the figure, the deadline is specified between when the request is made and when the delivery is made. In the second case, the only pick-up dead line is specified where the item needs to be picked up by the specified deadline. In the final case, the delivery deadline limits the time between when the item is picked-up and when the drop-off is made. We also assume that when a robot picks up or drops off an item at a location, the robot stays at the location for a variable but bounded service time.

Depending on the application a capacity of robot is limited to one or multiple items can be carried by a robot. Multiple items can be picked up and dropped off at multiple locations and can be redirected to pick-up new requested delivery while being dispatched for the previously assigned task. When multiple items are picked up and dropped off at one location, we assume that the service time at a location is independent of number of items. Also, we assume that the service time is not interruptible in that the robot cannot be redirected for new task while they stay at a location.

The total travel distance is limited by battery capacity. The battery model is introduced in the next subsection.

5.2.2 Mobile Robot Battery Model

We employ simple energy consumption model.

$$E_{motion}(d) = d * c_{motion} \tag{5.1}$$

for when a robot is in motion, and

$$E_{idle}(t) = t * c_{idle} \tag{5.2}$$

when a robot is not in motion, where t and d are time and distance respectively.

As a system parameter, the capacity of robots is given as $E_{capacity}$ and the charging

rate of batter, $r_{charging}$ is given.

$$t_{charging} = \frac{E}{r_{charging}} \tag{5.3}$$

where E is the amount of energy to be charged.

When multiple robots are required to charge at the same time, with single charging station, the robots have to wait their turn to charge. This overhead renders robot not operational longer than necessary to charge its battery and effectively reduces the operational number of robots. Therefore, reducing battery charging overhead is an important factor to be considered in this paper.

5.3 Path-based cost analysis for dispatch decision

5.3.1 Characteristic of a Delivery Request and a Delivery Path

t^n_{req}	request time
p^n	pick-up location
d^n	drop-off location
$t^n_{deadline}$	deadline

Table 5.1: A valid sequence of locations to visit by a robot R_j for two tasks

A delivery request, n, is defined by two locations, pick-up and drop-off locations, p^n and d^n, respectively. Depending on the application one of three deadlines discussed in the previous section is specified. In TABLE 5.1 notations used in the followings are summarized for a delivery request n. Its pick-up and drop-off locations, p^n and

d^n takes one of the specified locations.

	p^n			d^n
Q^n	a	b	c	d

Table 5.2: A pick-up and drop-off location of a new delivery request n is mapped to a path

When a robot is assigned to a single delivery request, a path, a sequence of locations, is decided. Even there are multiple paths between two locations, in the following discussion, the shortest distance is chosen for the path for the new request. In TABLE 5.2, the shortest path, a sequence of location, is mapped from the pick-up and the drop-off locations.

	p^1 d^1	p^2			p^3 d^2		d^3
P_j	a	b	c	d	e	f	c

Table 5.3: A remaining locations to visit by robot R_j is mapped to a static path

When a robot is assigned with more than two delivery tasks, a delivery path consists of multiple pick-up and drop-off locations as shown in TABLE 5.3, where three tasks are assigned to the robot. In the table, p^i and d^i stand for pick-up and drop-off location of a delivery request i. In the second row, the actual path the robot follows where the pick-up and drop-off locations are mapped to actual locations in the map. Note that in the actual path can contain same locations more than once.

With assumption that the service time at each location is independent of the number items to drop-off or pick-up at that location, with k locations with drop-off or pick-up tasks in a path the remaining time for the robot to finish the current path

can be obtained with current location of the robot.

$$t^j_{path} = \sum_k t_{service} + t^j_{travel}, \tag{5.4}$$

where $t^j_{dispatch}$ is the remaining travel distance to complete the current path and given as,

$$t^j_{travel} = \frac{d^j_{travel}}{v_j}, \tag{5.5}$$

where d^j_{travel} is the distance the robot travel to finish the current path and v_j is the speed of the robot. For a robot, not assigned with any task, the length of path, $t_{path} = 0$ and $t_{travel} = 0$. The location of robot and t_{path} and t_{travel} for the currently assigned path is known to the system at every sampling time. It is assumed that at every stop at pick-up or drop-off locations $t_{service}$ time is not known but its upper and lower bound is known. In the following discussion, it is assumed that $t_{service}$ time is randomly chosen between its upper bound and lower bound.

5.3.2 Analysis with path

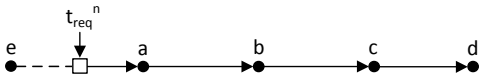


Figure 5-3: Remaining path for a Robot

Suppose that the distance of a path from the new request is d_p and upper bound of completion time of a path is t_p.

In this subsection we discuss robot's travel distance contributed when a new re-

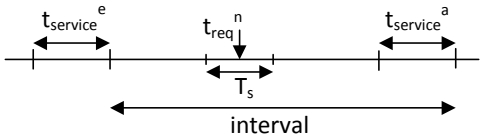


Figure 5-4: Robot is considered in between location e and a when the new request arrives after finishing service at e and before starting service at a

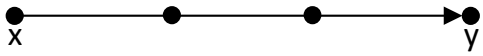
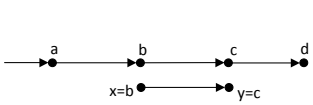


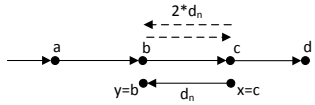
Figure 5-5: The path for a new request starting at x and ending at y.

quest is assigned to a robot that already has a scheduled path of locations to visit. Also, the affected tasks due to the new request in terms of additional delay is analyzed. In the following analysis it is assumed that a robot is heading to location a and the reaming path is $[a\ b\ c\ d]$. The pick-up and drop-off locations of a new request is shown as x and y in the following figures. Although only starting and ending position is shown for the new path is shown, it is also applicable when there are multiple locations between in the new path.

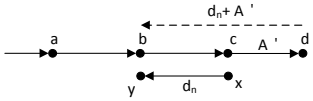
The first case we consider is when the path of new requested task is included in the path of a robot as shown in Fig. 5-6. There are three sub-cases depending on the direction of the new path and the sequence of completing tasks. In the first sub-case in Fig. 5-6(a) the direction of the new path and the current path coincides. In this case there is no robot travel distance is generated by a new request if the request is assigned to the robot as long as the deadline condition is satisfied. In Fig. 5-6(b) and Fig. 5-6(c), the new path is in reverse direction. When this new path is assigned to the robot, either the robot is finished the new path before completing the current path or finishing the current path before the new path. Newly generated robot travel



(a) Current path includes a new path with same direction.



(b) Current path includes a new path with reverse direction. Newly generated travel distance: $2 * d_n$.



(c) Current path includes a new path with reverse direction. Newly generated travel distance: $d_n + A'$

Figure 5-6: Case I: The remaining path for a robot includes the path for new task. Dashed line is newly contributed travel by the new task.

distance is shown as the dashed line. When $d_n > A'$ case (c) is chosen but, otherwise, (b). Here, A' is the distance between the pick-up location and the last position of the existing path following on the robot's current path. When the remaining path of R_j is P_j and the path from a new request is given as Q^n, the condition for this case is given as

$$\text{Case I}: \quad p^n, d^n \in P_j \tag{5.6}$$

Algorithm 2: Generated travel distance, d_g, determination for Case I

if exist_path$(P_j, p^n, d^n) =$ true **then**
  $d_g = 0$ (Case I(a))

else

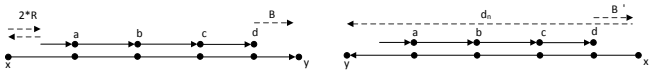
  if $2 * |Q^n| < |Q^n| + \text{dist}(P_j.\text{last\_loc}, d^n))$ **then**
    $d_g = 2|Q^n|$ (Case I(b))

  else
    $d_g = \text{dist}(P_j.\text{last\_loc}, d^n))$ (Case I(c))

  end

end

In Algorithm 2 the generated distance for Case I is determined. A predicate exist_subpath(p, x, y) gives true value if there is a path from x to y in the path p. $|Q^n|$ is the distance of the request path.



(a) Current path is included by a new path with same direction.

(b) Current path is included by a new path with reverse direction

Figure 5-7: Case O: When the current path for a robot is included by a path for new task. Dashed line is newly contributed travel by the new task.

In Fig. 5-7, a case O, where new request includes the existing remaining path of a robot. The condition for this case can be tested with,

$$\begin{aligned} \text{Case O}: \quad & p^n, d^n \notin P_j \ \wedge \\ & (p^n \gg (P_j)_s \gg d^n \vee d^n \gg (P_j)_s \gg p^n) \end{aligned} \tag{5.7}$$

where $p^n \gg (P_j)_s$ signifies the pick-up location is located before the current path and $(P_j)_s$ is the static path mapped from the current dynamic path of the robot.

Algorithm 3: Determine the relative location of pick-up or drop-off location relative to the remaining path

Input: $x = p^n$ or d^n

if path(l_j, x) contain path$(l_j, P_j.\text{last_loc})$ **then**
$P_j \gg x$

else
$x \gg P_j$

end

In Algorithm 3, for a given p^n or d^n how its relative location is obtained.

For R, the shortest between the current location of robot and the location of pick-up location of the new request is used.

$$R = \text{dist}(l_j, P_j.\text{first\_pos}) \tag{5.8}$$

where l_j is the location of robot j.

The distance from the last position of the current path to the pick-up and drop-off locations are given as,

$$\begin{aligned} B &= \text{dist}(\text{P}_\text{j}.\text{last\_pos}, \text{d}^\text{n}) \\ B' &= \text{dist}(\text{P}_\text{j}.\text{last\_pos}, \text{p}^\text{n}) \end{aligned} \tag{5.9}$$

Algorithm 4: Calculation of travel distance generated in Case O

In Case O:

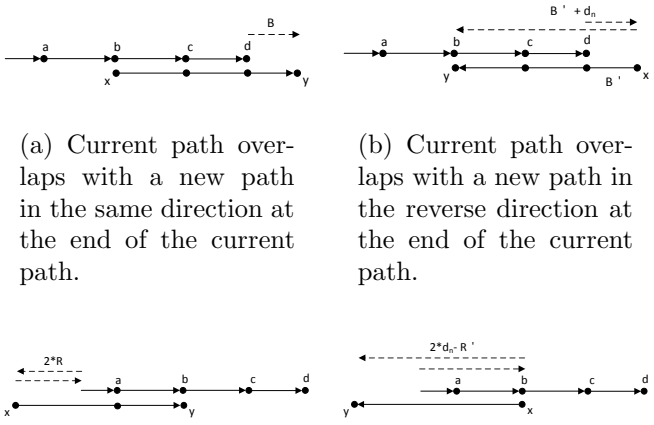
if $p^n \gg (P_j)_s \gg d^n$ **then**

$d = 2 * R + B$

else

$d = B' + |Q^n|$

end



(a) Current path overlaps with a new path in the same direction at the end of the current path.

(b) Current path overlaps with a new path in the reverse direction at the end of the current path.

(c) Current path overlaps with a new path in the same direction at the beginning of the current path.

(d) Current path overlaps with a new path in the reverse direction at the beginning of the current path.

Figure 5-8: Cases when the current path for a robot is overlapped with a path for new task. Dashed line is distance newly contributed by the new task.

In Fig. 5-8, four cases are illustrated when new path is partially overlapped with current remaining path for the robot.

$$\text{Case PO}: \ (p^n \in P_j \wedge d^n \notin P_j) \vee (p^n \notin P_j \wedge d^n \in P_j) \tag{5.10}$$

Algorithm 5: Calculation of travel distance generated in Case PO

In Case PO:

if $p^n \in P_j$ **then**

  if $P_j \gg d^n$ **then**
    $d = B$ (Case PO(a))

  else
    $d = 2 * |Q^n|'$ (Case PO(d))

  end

else

  if $P_j \gg p^n$ **then**
    $d = B' + |Q^n|$ (Case PO(b))

  else
    $d = 2 * R$ (Case PO(c))

  end

end

$$R' = \text{dist}(Q^n, l_j, p^n) \tag{5.11}$$

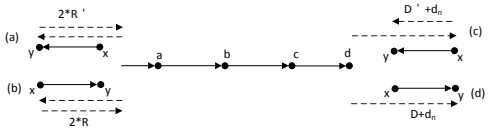


Figure 5-9: When the current path for a robot and a new path from the new request does not contain any common locations

In Fig. 5-9 the new path from the request does not overlap with the remaining path of the robot. As it can be shown in the figure the travel distance increases as the distance between two paths. The case when a robot is not assigned with any tasks

so does not have any remaining path is lower-right case in the figure where the travel distance generated is distance from the robot's location to the drop-off location of the task. This case is tested with,

$$\text{Case S}: \quad p^n, d^n \notin P_j \wedge (p^n, d^n \gg P_j \vee P_j \gg p^n, d^n) \tag{5.12}$$

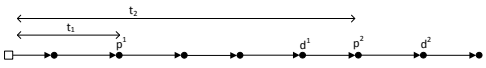
The distance between the last position and the pick-up and the drop-off locations are given as,

$$\begin{aligned} p^n \gg d^n \gg P_j &\rightarrow 2 * R' \\ d^n \gg p^n \gg P_j &\rightarrow 2 * R \\ P_j \gg p^n \gg d^n &\rightarrow B' + |Q^n| \\ P_j \gg d^n \gg p^n &\rightarrow B \end{aligned} \tag{5.13}$$

In TABLE 5.4 all possible cases between a new request and the remaining path of of robot is summarized. In all cases, when the direction of the path of a new request is as same direction as an existing path, the generated distance is smaller. For a given robot with its remaining path and a new task, the case can be determined and the generated distance can be calculated. The only ambiguity is between the case I(a) and I(b) in the table, where d_n and A needs to be compared.

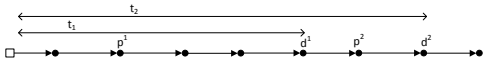
Case	Sub-cases	Distance
I	(a)	0
	(b)	$2 * d_n$
	(c)	$d_n + A'$
O	(a)	$2 * R + B$
	(b)	$R' + B'$
PO	(a)	B
	(b)	$d_n + B'$
	(c)	$2 * R$
	(d)	$2 * d_n$
S	(a)	$2 * R'$
	(b)	$2 * R$
	(c)	$B' + d_n$
	(d)	B

Table 5.4: Summary of distance generated from a new request with an existing path. I: included case, O: included case, PO: partially overlapped case and S: separated case



(a) The affected range of locations when pick-up deadline is specified

(b) The affected range of locations when overall deadline is specified



(c) The affected range of locations when delivery deadline is specified

Figure 5-10: Different affected range of locations when different types of deadlines are specified

5.3.3 Path Deadline

In Fig. 5-10, the affected range of nodes are shown when different types of deadlines are given. When a new request modify the existing path for a robot, the deadline conditions for each node needs to be checked. For each location, the system maintains a set of tasks that are either pick-up or drop-off point on that location.

task	Robot	request time	deadline	deadline type
p^a	i	t^a_{req}	$t^a_{deadline}$	p, o
p^b	j	t^b_{req}	$t^b_{deadline}$	p, o
d^c	k	t^c_{req}	$t^c_{deadline}$	d

Table 5.5: Node Table

For every node in the affected range of path, when a new request modifies the existing path, for a new path from the robot to the node, the path time is estimated with (5.4). With t_{req}, $t_{deadline}$ and $t_{current\_time}$, the remaining deadline is calculated and which should be larger than estimated path time for the new path to be a valid one.

The remaining deadline of its task is calculated when a new task is requested. If additional time generated by a new task that is assigned to the current path of a robot

Algorithm 6: Shortest new travel based dispatch for a new request, n

Input: $\{P_j\}$: the remaining paths for all robots

Input: Q^n: path for a new request, n

foreach P_j **do**

Decide case and calculate generated travel distance(d_j)

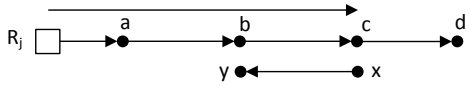
Check deadline condition for the new and existing tasks

end

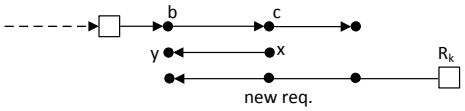
Choose robot with minimum d_j without deadline miss

5.3.4 Enhancement to shortest travel dispatch strategy

In this section we introduces two enhancements to the basic mechanism. In Fig. 5-



(a) A new task is assigned to R_j



(b) Before R_j reaches the pick-up location of the new request another robot R_k can be assigned with the task with shorter generated travel distance

Figure 5-11: Case where delayed scheduling can generate less travel distance

11(a), robot R_j is assigned with a new task which generates shortest travel distance. However, before the robot pick up the item at location c, other request can be generated which generates even shorter travel distance contributed by the new task. In

order to exploit this advantage the task of which item has not been picked up is kept in pending state with calculated travel distance. In circular topology the only possible case that will make generated travel distance is shorter is by the robot moving in reverse direction.

Delayed start strategy: related to pick-up location request.

5.3.5 Battery-level aware dispatch strategy

As additional constraint when assigning a new task to a robot, the battery level of the robot is enough to go back to the charging station. Battery usage of a path is given as,

$$E_{path} = d_{travel} * c_{motion} + t_{path} * c_{idle} \tag{5.14}$$

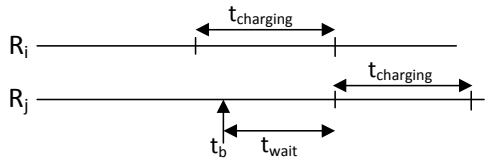


Figure 5-12: Delay at charging station with uncontrolled charging station access

In Fig. 5-12 when robot R_j arrives to the charging station R_i is being charged and R_j is forced to wait until its turn. The objective is minimize t_{wait} by allowing robots to charge battery before its exhaustion in a controlled way. Another value to minimize is the distance that a robot has to travel to charge the battery without executing tasks.

When a new path is generated the following condition should be hold with inclu-

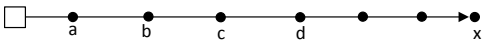


Figure 5-13: Valid path with battery level consideration

sion of travel to the charging station,

$$E_{path} < E_{current\_battery\_level} \tag{5.15}$$

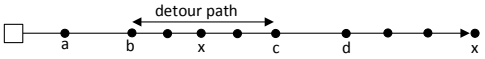


Figure 5-14: Alternative path with charging between task execution

In Fig. 5-14 the alternative path for a robot to execute a path where the robot recharges battery between visiting path locations. The alternative path should not break existing deadline conditions. Total time to detour is given with,

$$t_{detour} = t_{wait} + \frac{d_{detour}}{v_r} + t_{charging} = \frac{d_{tour}}{v_r} + \frac{E}{r_{charging}}, \tag{5.16}$$

where E, the amount of energy to be charged, is enough for the remaining path to be completed including come back to the charging station. The t_{wait} depends on the current scheduled battery charges of the charging station.

5.4 Simulation & Evaluation

In this section the proposed dispatch strategy, the shortest travel distance dispatch strategy(ST), is compared with a basic dispatch strategy where a robot is chosen for

Number of Locations	8
Request Rate	$0.5 \sim 15.0$ (requests/min)
Number of Robots	8
Distance between Locations	10 (meter)
Robot Speed	2 (m/s)
Service Time	120 (sec)
Request Deadline	1000 (sec)

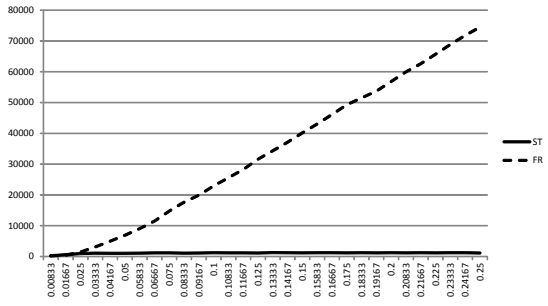
Table 5.6: The summary of parameters used in the evaluation

a task where the initial dispatch distance from the robot's location and the pickup location is minimum - First Robot Dispatch(FR).

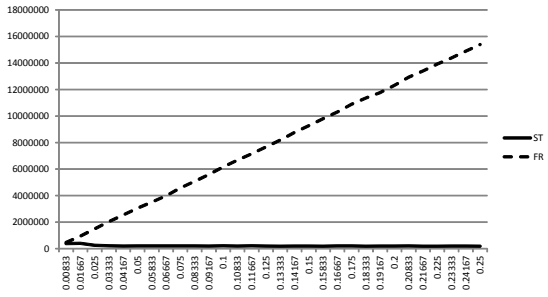
In TABLE 5.6 the parameters used in the evaluation is summarized. Where the topology is chosen circular as shown in Fig. 5-1(b). In the evaluation the arrival rate is varied and the performances of the strategy, average delay, travel distance and block rate, are compared with deadline specified and without deadline. The likelihood of task arriving at each node is assumed to be uniform. The number of items a robot can carry is assumed to be unlimited.

In Fig. 5-15, the average delay and the total travel distance of robots are compared when shortest travel dispatch strategy(ST) and first robot dispatch strategy(FR) is used. With FR as the rate of tasks increases both the delay and travel distance is linearly increased. However with ST, the delay and the travel distance are not affected with the increasing incoming tasks. It is due to that with more requests there are more tasks that can be combined with existing travel path of the robot.

In Fig. 5-16 the average delay, the blocking rate of the requested tasks and the travel distance of the robots are compared when the shortest travel dispatch strategy(ST) and the first robot dispatch strategy(FR) is used. In Fig. 5-16(a) the

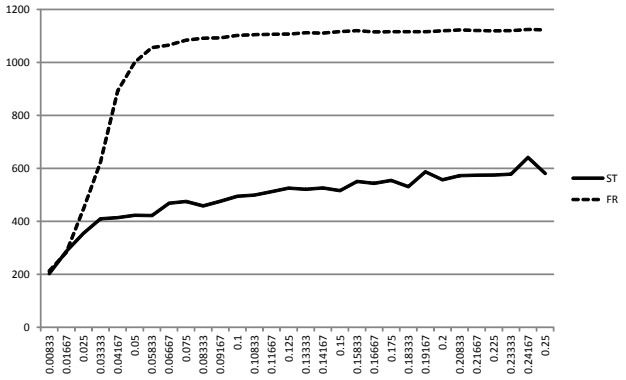


(a) Average Delay (sec)

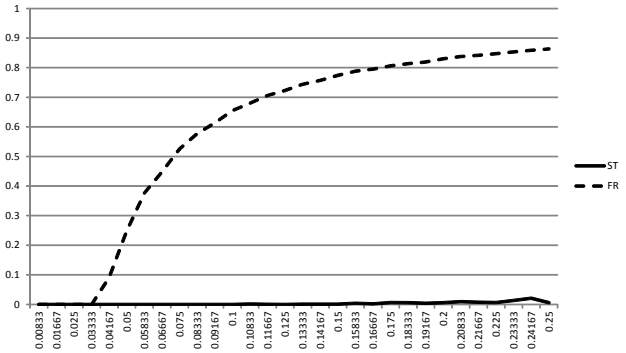


(b) Travel Distance (10^{-1} meter)

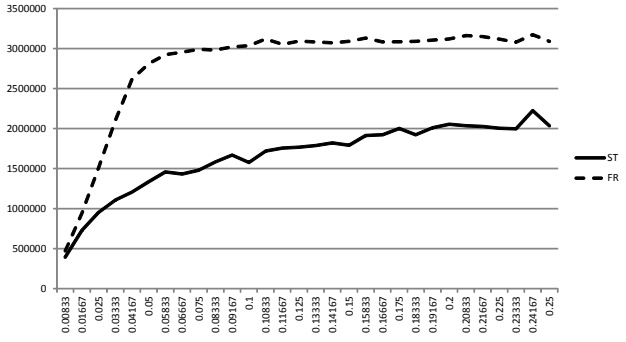
Figure 5-15: Comparison of shortest travel distance dispatch strategy(ST) and first robot dispatch strategy(FR) when deadline is not specified



(a) Average Delay (sec)



(b) Blocking Rate



(c) Travel Distance (10^{-1} meter)

Figure 5-16: Comparison of shortest travel distance dispatch strategy(ST) and first robot dispatch strategy(FR) when deadline is specified to 1200 (sec)

average deadline is limited with specified deadline 1200 (sec) when FR is used. The average delay of the tasks when ST is used increases with the incoming request rate but it increases much slowly as shown in Fig. 5-16(a). In Fig. 5-16(b), it is shown that task blocking rate increases rapidly with FR. The total travel distance by robots with FR also limited because of the specified deadline, but ST it increases slowly.

5.5 Conclusion and Future Work

The path based cost analysis for scheduling decision making in multi robot dispatch environment is presented and an effective dispatch strategy for delivery requests are proposed. It is shown that a predefined path dispatching strategy, considering pickup and delivery locations of the new task and the existing path of the robot, is highly effective. The on-going work is to extend this path based analysis to the application of dispatching to dynamically moving objects and dispatch strategy in such environments.

Chapter 6

Conclusion and Future Research

6.1 Conclusion

This dissertation contains work using divisible load theory in multiple areas. The first topic deals with a practical situations when large amounts of data are generated from experiments or the simulation of large scientific projects and such data needs to be processed without storing it. We derive the capacity of a computing site composed of many computing processors connected with a bus network. The derivation is based on the optimal criterion of divisible load theory. The numerical example is simulated and presented. The second topic deals with an operation point of view of divisible loads. Flat file databases are assumed to have the divisible load property and the closed form solutions of the expected search time of kth signature is derived. With prior knowledge of the distribution of signature location, a procedure to speed up the signature search time is presented. Based on the proposed procedure, a simulation result is presented.

6.2 Future research

In the second chapter of this report, signature search time in flat files is examined, where data is modeled as divisible load and no further assumption is made on data except the distribution of the location of signature in the load. In practical environment, after initial processing of raw data, the data is usually sorted or/and indexed for speeding up the operation that may be performed later. For example, large amounts of structured data of such database records are well indexed in relational database system. Also, in most cases, instead of matching a single signature in data, relations between the data is sought, a join operation for example. When such operations need to be performed on a distributed computing processor, the characteristic of operation and the characteristic of arrangement of data records will significantly impact the performance of the operation because of the communication cost incurred by the non-negligible size of the intermediate or the final result.

As a motivational example, we consider the join operation of typical relational database systems. When a single record needs to be found from a database table and a field that ought to be matched is indexed, it can be considered as a special case where the distribution of the location of signatures is known as presented in Chapter 2. When the field to be searched are the combination two fields from different tables, they need to be joined and need to be scanned. The basic approach of such operations is an inner-loop styled operations. In the inner-loop join operation, each record from one table is combined with all the records from the other table, and the combination of the two fields are matched against. When the number of data records in a table

is very big compared to the size of individual record, we can model the records from two database tables as two divisible loads. In such an operation, which is different from the single load case, one load needs to be transferred to each processor while the other load can be partitioned. When the number of records of two tables are of the size of M and N, the operation time of the inner-loop join operation takes $O(MN)$ time. When there are K processors, the total time of operation will be reduced to, $O(M * N/K)$, when communication time is ignored. However, when a distributed system connected with a communication channel is considered, this maximum speed-up cannot be achieved. Moreover, as the large literature of DLT has shown, a naive approach to distribute the load will diminish the advantage of multiple processors in a significant way. Also, when the communication cost is not negligible in networked environments, the strategy as to which load to distribute will depend on the relative size of the load and when multiple records need to be selected, as in typical database operations, the report time also affects the finish time. In the future work, the operation time and load distribution strategy will be examined in such an operation.

Once the basic analysis for the join operation using divisible load theory is established, the work can be further extended to more complicated operations. For example, modern database system incorporates hashed join operation to avoid repeated heavy computational cost of the inner join operation. Several such typical operations can be examined and analyzed using divisible load theory for distributed computing environment with non-negligible communication cost. In [61], several database operations are described. The operations can be categories in two dimension. The first dimension of categorization consists of sequential sub-operations where the result of

the one stage of sub-operation is used in the subsequent sub-operation so that parallel processors cannot be utilized to shorten the operation time. The second dimension is where the load can be partitioned into parallel sub-operations, where parallelism of multiple processors can be explored to speed up the operation time. In this categorization, the communication cost plays a significant role at two points. Firstly, when the result of one sub-operation is subsequently used by another operation, communicating the intermediate result can be significant and needs to be considered when distributing the loads to be processed. Secondly, when partitioning the load for parallel processing, without a smart scheduling strategy, the communication cost to deliver the partial load to the processors can be reduced by taking the distribution of the records into account. In our future work, through the analysis and simulation, general guidelines of distributing strategy will be attempted to be established. Such guidelines are expected to be dependent upon the statistics of the data records, the size of the final or intermediate result and, most importantly, the characteristic of the operation.

Bibliography

[1] K. Ko and T. G. Robertazzi. Signature search time evaluation in flat file databases. *IEEE Transactions on Aerospace and Electronic Systems*, 44(2):493–502, April 2008.

[2] J. T. Hung and T. G. Robertazzi. Distributed scheduling of nonlinear computational loads. *2003 Conference on Information Sciences and Systems*, March 2003. The Johns Hopkins University.

[3] V. Bharadwaj, D. Ghose, and T.G. Robertazzi. A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–18, Jan 2003. in special issue of Cluster Computing on Divisible Load Scheduling.

[4] V. Bharadwaj, D. Ghose, and T.G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *in special issue of Cluster Computing on Divisible Load Scheduling*, spring 2003. D. Ghose and T.G. Robertazzi editor.

[5] The Large Hardon Collider : http://lhc.web.cern.ch/lhc.

[6] Search for EXtraterrestrial Intelligence(SETI): http://setiathome.berkeley.edu.

[7] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press (now distributed by Wiley), Los Alamitos, CA, USA, 1996.

[8] S. Bataineh and T. G. Robertazzi. Bus oriented load sharing for a network of sensor driven processors. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1202–1205, 1991.

[9] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: Results and open problems. *IEEE Transactions on Parallel and Distributed Systems*, 16(3), 2005.

[10] G. D. Barlas. Collection aware optimum sequencing of operations and closed form solutions for the distribution of divisible load on arbitrary processor trees. *IEEE Transactions on Parallel and Distributed Systems*, 9(5):429–441, May 1998.

[11] J. Blazewicz and M. Drozdowski. Scheduling divisible jobs on hypercubes. *Parallel Computing*, 21(12):1945–1956, 1995.

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

[13] K. Ko and T.G. Robertazzi. Scheduling in an environment of multiple job submissions. In *Proceedings of the 2002 Conference on Information Sciences and Systems*, Princeton, NJ, USA, 2002.

[14] V. Bharadwaj and G. Barlas. Efficient scheduling strategies for processing multiple divisible loads on bus networks. *Journal of Parallel and Distributed Computing*, 62:132–151, 2002.

[15] H.M. Wong, D. Yu, B. Veeravalli, and T.G. Robertazzi. Data intensive grid scheduling: Multiple sources with capacity constraints. In *Proc. of the IASTED International Conference on Paralle and Distributed Computing and Systems*, Los Angeles, USA, Nov 2003. IEEE.

[16] M. Moges, D. Yu, and T.G. Robertazzi. Divisible load scheduling with multiple sources: Closed form solutions. In *Proceedings of the 2005 Conference on Information Sciences and Systems*, The Johns Hopkins University, Baltmore, MD, USA, March 2005.

[17] V. Bharadwaj, D. Ghose, and V. Mani. Optimal sequencing and arrangement in distributed single-level tree networks with communication delays. *IEEE Transactions on Parallel and Distributed Systems*, 5:968–976, 1994.

[18] S. Charcranoon, T. G. Robertazzi, and S. Luryi. Load sequencing for a parallel processing utility. *Journal of Parallel and Distributed Computing*, 64:29–35, 2004.

[19] S. Bataineh and T. G. Robertazzi. Performance limits for processor networks with divisible jobs. *IEEE Transaction on Aerospace and Electronic Systems*, 33:1189–1198, 1997.

[20] J. Blazewicz and M. Drozdowski. The performance limits of a two dimensional network of load sharing processors. *Foundations of Computing and Decision Sciences*, 21(1):3–15, 1996.

[21] S. Suresh, V. Mani, and S. N. Omkar. The effect of start-up delays in scheduling divisible loads on bus networks: An alternate approach. *Computers & Mathematics with Applications*, 46(10-11):1545 – 1557, 2003.

[22] V. Bharadwaj, H. F. Li, and T. Radhakrishnan. Scheduling divisible loads in bus networks with arbitrary processor release times. *Computers & Mathematics with Applications*, 32(7):57 – 77, 1996.

[23] Bharadwaj Veeravalli and Jingnan Yao. Divisible load scheduling strategies on distributed multi-level tree networks with communication delays and buffer constraints. *Computer Communications*, 27(1):93 – 110, 2004.

[24] M.A. Roth, S.A. Ruberg, and B.L. Eldridge. Database management: the heart of integrated avionics. *Aerospace and Electronics Conference, 1993. NAECON 1993., Proceedings of the IEEE 1993 National*, pages 535–541 vol.1, May 1993.

[25] Ching-Shan Peng and Kwei-Jay Lin. A performance study of the concurrency control algorithms for real-time avionics databases. *Digital Avionics Systems Conference, 1997. 16th DASC., AIAA/IEEE*, 1:1.2–23–30 vol.1, Oct 1997.

[26] Magnus Lübeck, Dirk Geppert, Krzysztof Nienartowicz, Marcin Nowak, and Andrea Valassi. An overview of a large-scale data migration. In *MSS '03: Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, page 49, Washington, DC, USA, 2003. IEEE Computer Society.

[27] S. Christodoulakis. Implications of certain assumptions in database performance evauation. *ACM Trans. Database Syst.*, 9(2):163–186, 1984.

[28] H. K. Harter. *Ordering Statistics and Their Use in Estimation and Testing, vols. 1 and 2.* Washington D.C.: Superintendent of Documents, U.S. Government Printing Office, 1969.

[29] Y. C. Cheng and T. G. Robertazzi. Distributed computation with communication delays. *IEEE Transactions on Aerospace and Electronic Systems*, 24(6):700–712, 1988.

[30] Y. C. Cheng and T. G. Robertazzi. Distributed computation for tree networks with communication delays. *IEEE Transactions on Aerospace and Electronic Systems*, 26:511–516, 1990.

[31] R. Agrawal and H.V. Jagadish. Partitioning techniques for large-grained parallelism. *Computers and Communications, 1988. Conference Proceedings., Seventh Annual International Phoenix Conference on*, pages 31–38, Mar 1988.

[32] T. Robertazzi. Ten reasons to use divisible load theory. *Computer*, 36(5):63–68, May 2003.

[33] B. Veeravalli and Wong Han Min. Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times. *Parallel and Distributed Systems, IEEE Transactions on*, 15(3):273–288, March 2004.

[34] Bharadwaj Veeravalli, Xiaolin Li, and Chi Chung Ko. On the influence of start-up costs in scheduling divisible loads on bus networks. *IEEE Trans. Parallel Distrib. Syst.*, 11(12):1288–1305, 2000.

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

[36] S. Charcranoon, T.G. Robertazzi, and S. Luryi. Parallel processor configuration design with processing/transmission costs. *IEEE Transactions on Computers*, 49(9):987–991, Sep 2000.

[37] Jia Jingxi, B. Veeravalli, and D. Ghose. Adaptive load distribution strategies for divisible load processing on resource unaware multilevel tree networks. *Computers, IEEE Transactions on*, 56(7):999–1005, July 2007.

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

[39] Yeim-Kuan Chang, Jia-Hwa Wu, Chi-Yeh Chen, and Chih-Ping Chu. Improved methods for divisible load distribution on k-dimensional meshes using multi-installment. *IEEE Trans. Parallel Distrib. Syst.*, 18(11):1618–1629, 2007.

[40] Xiaolin Li, B. Veeravalli, and Chi Chung Ko. Divisible load scheduling on a hypercube cluster with finite-size buffers and granularity constraints. *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 660–667, 2001.

[41] Jingnan Yao and Bharadwaj Veeravalli. Design and performance analysis of divisible load scheduling strategies on arbitrary graphs. *Cluster Computing*, 7(2):191–207, 2004.

[42] Veeravalli Bharadwaj and Gerassimos Barlas. Scheduling divisible loads with processor release times and finite size buffer capacity constraints in bus networks. *Cluster Computing*, 6(1):63–74, 2003.

[43] M. Drozdowski and M. Lawenda. Multi-installment divisible load processing in heterogeneous distributed systems. *Concurrency and Computation: Practice and Experience*, 19(17):2237–2253, 2007.

[44] Zeng Zeng and Bharadwaj Veeravalli. Divisible load scheduling on arbitrary distributed networks via virtual routing approach. In *ICPADS '04: Proceedings of the Parallel and Distributed Systems, Tenth International Conference*, page 161, Washington, DC, USA, 2004. IEEE Computer Society.

[45] D. Ghose, Hyoung Joong Kim, and Tae Hoon Kim. Adaptive divisible load scheduling strategies for workstation clusters with unknown network resources. *Parallel and Distributed Systems, IEEE Transactions on*, 16(10):897–907, Oct. 2005.

[46] He Li, Guangzhong Sun, and Yinlong Xu. Distributed scheduling strategies for processing multiple divisible loads with unknown network resources. *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 824–829, Sept. 2007.

[47] Maciej Drozdowski and Pawel Wolniewicz. Experiments with scheduling divisible tasks in clusters of workstations. In *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 311–319, London, UK, 2000. Springer-Verlag.

[48] T. G. Robertazzi. *Networks and Grids: Technology and Theory*. Springer, 2007.

[49] M. Drozdowski. *Scheduling for Parallel Processing*. Springer, New York, 2009.

[50] H. Drozdowski, A. Legrand, and Y. Robert. *Parallel Algorithms*. CRC Press, Boca Raton, Florida, 2009.

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

[52] Bharadwaj Veeravalli and Surendra Ranganath. Theoretical and experimental study on large size image processing applications using divisible load paradigm on distributed bus networks. *Image and Vision Computing*, 20(13-14):917 – 935, 2002.

[53] V. Bharadwaj, Xiaolin Li, and Chi Chung Ko. Efficient partitioning and scheduling of computer vision and image processing data on bus networks using divisible load analysis. *Image and Vision Computing*, 18(11):919 – 938, 2000.

[54] Teo Tse Chin, Bharadwaj Veeravalli, and Jingxi Jia. Handling large-size discrete wavelet transform on network-based computing systems – parallelization via divisible load paradigm. *Journal of Parallel and Distributed Computing*, 69(2):143 – 152, 2009.

[55] S. K. Chan, V. Bharadwaj, and D. Ghose. Large matrix-vector products on distributed bus networks with communication delays using the divisible load paradigm: performance analysis and simulation. *Mathematics and Computers in Simulation*, 58(1):71 – 92, 2001.

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

[57] M. Drozdowski and M. Lawenda. The combinatorics in divisible load scheduling. *Foundations of Computing and Decision Sciences*, 30(4):297–308, 2005.

[58] D. Bitton, D. DeWitt, and C. Turbyfill. Benchmarking database systems: A systematic approach. In *Proceedings of the 9th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Florence*, 1983.

[59] Kwangil Ko and T.G. Robertazzi. Equal allocation scheduling for data intensive applications. *Aerospace and Electronic Systems, IEEE Transactions on*, 40(2):695–705, April 2004.

[60] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. New York Cambridge University Press, 3 edition, 2007.

[61] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, 2008.