Scheduling divisible loads for parallel and real time systems, distributed networks, and computational grids

A Dissertation Presented

by

Taeyoung Lim

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

December 2007

Stony Brook University

The Graduate School

Taeyoung Lim

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

Milutin Stanacevic,
Assistant Professor, Department of Electrical & Computer Engineering

Hussein Badr,
Associate Professor, Department of Computer Science

This dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Scheduling divisible loads for parallel and real time systems, distributed networks, and computational grids

by

Taeyoung Lim

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

2007

In this dissertation, four scheduling problems in parallel video processing systems, real-time systems, networks, and computational grids are considered.

Communication delay in a processor network is very critical to the throughput of a parallel video processing system. The interaction of communication and computation is examined here in a number of contexts. First, a simultaneous distribution and collection method (SD) from the root processor to children processors via a multi-port switch network is proposed. For the proposed mechanism, we analyze the video encoding time and derive a closed-form solution for a multi-port star interconnect network. The results show that the total encoding time is significantly faster than a previous method, Parallel Interlaced (PI) [1], based on a bus network. In addition, we achieve scalability in terms of the number of processors because of the concurrent communication.

The deleterious impact of communication on computation for computers is one of the factors that affect performance of computers in a network. A scheduling method considering this interference of communication on computation is proposed in detail and analyzed here from the perspective of divisible load theory in heterogeneous networks and grids. Each processor is divided into two virtual processors with different computing speeds according to the degree of overlapping communication. These two virtual processors are used to obtain one equivalent computing speed. Through this process we obtain the closed-form solution for the processing time considering the effect of communication on computation. In addition, interference aware scheduling is extended from sequential distribution to simultaneous distribution and applied to parallel video processing. A concurrent scheduling method considering communication interference (IA-COMP) reflects more realistic and accurate results in this specific application.

It will be increasingly common that multiple source nodes originate workload to sink nodes in very large heterogeneous networks. A multi-source scheduling scheme through network partitioning is proposed. All sink nodes are involved in only one partition associated with a single source node. In partitioning the network, the sum of computing speeds of the sink nodes in each partition should be similar to one another for the initial network partition. Each partition is evaluated by running an optimal single-source sequential load distribution scheduling method whenever network partitioning is repeated. After every evaluation, new partitions could be constructed by transferring sink nodes from one partition to another or rearranging the sequence of sink nodes receiving load from the source node within a partition. We iterate this partitioning and evaluation via a genetic algorithm until a globally near-optimal solution is approached or obtained.

It has been increasingly important to provide performance guarantees to deadline-constrained jobs originating from large scale experiments. To satisfy the deadline of workloads, computing and communication capability should be guaranteed and each workload should be estimated through a schedulability test before submission. In doing so, divisible load theory (DLT) has recently been extended with real-time characteristics, where the job with the earliest deadline is the first scheduled. However, such a method still is limited to only a homogeneous cluster environment. In this paper, scheduling heuristics involving network partitioning are proposed for large scale heterogeneous Grid/cluster systems. The minimum number of nodes obtained from a homogeneous model is used in two-level schedulability test of a job in the original heterogeneous system. The entire network is fragmented into small partitions, where the minimum number of children nodes are selected to be just large enough to satisfy the deadline of each job for each partition. Intensive simulations show that our proposed real-time scheduling method via DLT provides not only feasible solutions applicable to a heterogeneous system, but shows also good performance than in terms of the rejection ratio of jobs.

All glory to my Lord, Jesus Christ

To my late mother, Bunja Park and my father, Kyusu Lim,

To my wife, Eunjung Yoon and two sons, Konhah and Chanhah

Contents

List of Figures x

List of Tables xiii

Acknowledgements xiv

1 Introduction 1

1.1 Background 1

1.2 Motivation 3

1.3 Contribution 4

1.4 Outline 6

2 Concurrent Scheduling 9

2.1 Concurrent Communication In Parallel Video Processing 12

2.1.1 Interconnect Topology 12

2.1.2 Scheduling Scenario 16

2.1.3 Notation 17

2.2 Simultaneous Distribution Scheduling (SD) 18

2.2.1 No computation on the root processor (SD-NO) 18

2.2.2 Computation on the root processor (SD-COMP) 21

2.3 Performance Analysis And Comparison 26

2.3.1 Speedup of the SD-NO method 26

2.3.2 Speedup of the SD-COMP method 27

2.4 Conclusion 29

3 Interference Aware Scheduling **31**

3.1 Sequential Distribution Model and definition 32

3.1.1 Sequential distribution model for a heterogeneous tree 32

3.1.2 Notation 33

3.2 Interference aware scheduling (IA) 34

3.2.1 Timing Diagram for Sequential Distribution 34

3.2.2 Analytical solution 37

3.3 Numerical Solution and comparison 44

3.3.1 Numerical Solution Parameters 44

3.3.2 Numerical Solution Results 44

3.4 Interference Aware Scheduling In Parallel Video Processing 47

3.4.1 Notation for the IA method 47

3.4.2 Interference aware scheduling (IA-COMP) 48

3.4.3 Comparison of the SD-COMP and IA-COMP method 52

3.5 Conclusions 55

4 Multi-source scheduling **56**

4.1 Motivation 56

4.2 Multi-source scheduling in a large heterogeneous network 59

4.2.1 Problem formulation and definition 59

4.2.2 Two source scheduling 61

4.2.3 Network partitioning 64

4.2.4 Load distribution sequencing 66
4.3 Multi-source scheduling via genetic algorithm 71
4.3.1 Genetic algorithm heuristics 71
4.3.2 Network partitioning via crossover operator 73
4.3.3 Load sequencing via mutation operator 75
4.4 Performance analysis and comparison 78
4.5 Conclusion 80
5 Real-time scheduling 82
5.1 Models and problem description 86
5.1.1 Models and definition 86
5.1.2 Problem description 88
5.2 Deadline-constrained scheduling in a heterogeneous model 89
5.2.1 Application specific scheduling in a heterogeneous model . . . 89
5.2.2 Deadline-constrained network partitioning 93
5.3 Performance analysis and comparison 100
5.3.1 Simulation parameters 100
5.3.2 Performance analysis 100
5.4 Conclusion 107
6 Future Research and Conclusion 109
6.1 Conclusion 109
6.2 Future research 111

List of Figures

1-1 A multi-source heterogeneous Grid/cluster network configuration of the ATLAS project. 3

2-1 Block diagram for multi-port interconnection network. 12

2-2 The simultaneous load distribution in a tree network. 16

2-3 The timing diagram for the SD-NO method. 20

2-4 The timing diagram for the SD-COMP method. 21

2-5 The finish time versus the number of processors for SD-NO, PI and PR load scheduling methods. 26

2-6 The finish time versus the number of processors for SD-COMP, SD-NO, PI, and PR load scheduling methods on a homogeneous network. . . . 28

3-1 Interference aware sequential load distribution in a tree topology. . . 33

3-2 Timing diagram for IA (Interference Aware) sequential distribution and simultaneous start. 35

3-3 The equivalent processor concept for IA (interference aware) scheduling. 40

3-4 Timing diagram of the equivalent processor for Interference Aware sequential distribution and simultaneous start scheduling. 41

3-5 The IA (Interference Aware) scheduling versus the SD (Sequential Distribution) Scheduling. 45

3-6 The IA (Interference aware) scheduling method versus than the SD (Sequential Distribution) method in terms of communication speed. . 46

3-7 The timing diagram for the IA-COMP(Interference Aware scheduling with computation the root processor) method. 50

3-8 The SD-COMP scheduling method versus IA-COMP method. 53

4-1 Multi-source heterogeneous network 60

4-2 Two partitions should be created so that the sum of the computing speed ($\frac{1}{w}$) of both partitions can be similar each other if possible. . . 62

4-3 Timing diagram for multi-source scheduling with sequential distribution in a network with two source and seven sink nodes. $T_{f,1}$ and $T_{f,2}$ are the finish times and T_w is the waiting time. 63

4-4 Flowchart for multi-source scheduling via genetic algorithm in grid network 67

4-5 The network partitioning via the crossover operator. 74

4-6 Node sequencing via the mutation operator. 76

4-7 The near-optimal solutions for the finish time, $T_{f,m}$ 78

4-8 The near-optimal solutions for the waiting time, T_w 80

5-1 A heterogeneous Grid/cluster system example, TeraPaths network configuration of the ATLAS project. 84

5-2 A heterogeneous model with deadline-constrained jobs. 90

5-3 Timing diagram for a heterogeneous model. 91

5-4 Transformation of a heterogeneous network into a homogeneous network 93

5-5 Deadline-constrained network partitioning in a heterogeneous tree network 96

5-6 Timing diagram for deadline-constrained network partitioning in a heterogeneous tree network. 97

5-7 Two-level schedulability test in a heterogeneous model. 98

5-8 The finish time for heterogeneous and transformed homogeneous models. 103

5-9 The rejection ratio for heterogeneous and transformed homogeneous models with the system load. 103

5-10 The rejection ratio for heterogeneous and transformed homogeneous models with inter-arrival time ($1/\lambda$). 104

5-11 EDF versus FIFO in a heterogeneous and homogeneous model. 105

5-12 The rejection ratio of application specific heterogeneous systems. . . . 106

5-13 Minimum nodes network partitioning in a heterogeneous model. . . . 107

6-1 Multi-source real-time scheduling scenario 111

List of Tables

2.1 The Properties of topologies. $'n'$: a dimension of Hypercube, $'p'$: the number of nodes along one edge of Mesh and Torus, $'h'$: the height of the binary tree. (*K.Hwang et al.* [2], *Duato et al.* [3], *http : //www.netlib.org/benchmark/top*500 [4]). 13

3.1 The ratio of SD-COMP to IA-COMP method in terms of finish time for the number of processors. 54

3.2 The ratio of SD-COMP to IA-COMP method in terms of finish time with communication speed decreased. 54

5.1 The notations for the algorithms tested in the simulation. 101

Acknowledgements

With the help of my wonderful advisor, Professor Thomas G. Robertazzi, I can stand here and express this thanks now. He has been considerate in inspiring me to investigate new problems as well as scrupulous in proofreading all of the papers including the dissertation. He has shown the role model to be followed as a real scholar and researcher.

I would like to thank the committee members of the final defense, Professor Sangjin Hong, Hussein Badr, and Milutin Stanacevic for their invaluable comments on the this dissertation. Especially, the assistance of Professor Hong is very appreciated. I am thankful to Professor Petar M. Djurić for providing facilities and to the colleagues in the COSINE Laboratory and to the Korean colleagues in the department of Electrical and Computer Engineering, Stony Brook University. The assistance of Dr. Jason Hung in developing equations and Kyoung-su Park in programming experiments is very helpful.

My thanks should go to Dr. Dantong Yu and Dr. Dimitrios Katramatos in the Grid Group, BNL(Brookhaven National Laboratory). Through the TeraPaths project I has involved in BNL, my works are truly motivated and get more fruitful in the last two main chapters.

I am also grateful to the current and old members who I has met in the Nanume church, especially to the pastor, Tae Jun Suk, elders and deacons and small group members. They have prayed for me every turning point of study abroad that I has been either by the rivers of water or in the valley of the shadow.

In general, I would like to thank my family for love and prayer during the Ph.D study. Especially, my late mother, Bunja Park poured out her love to my spirit and from her I inherited my faith in God. My father, Kyusu Lim and the parents-in-law, Kyungsim Park and Yeosung Yoon have been praying for my study day and night. I also want to share the joy with the family members and friends, especially the sister, Miae Lim.

Last, but not least, I must give special thanks to my wife, Eunjung Yoon and two sons, Konhah and Chanhah who have gone through this challengeable journey with overflowing love and confidence and made me happy all the way.

December 2007

Stony Brook, NY

Taeyoung Lim

The gospel to the poor;

To heal the brokenhearted.

(Luke 4:18,19)

Chapter 1

Introduction

1.1 Background

In this dissertation, four outstanding scheduling problems and their performance modelings are considered.

First, in parallel video processing, various scheduling algorithms to assign video frames to multiple processors have been presented to find both the maximum processing throughput and I/O utilization. Load (video frame) partition schemes through DLT (divisible load theory) are shown to obtain a good performance in parallel video encoding [5]. However the scheduling algorithms based on sequential communication have inherent limitations on communication in terms of throughput and the optimal number of processors. Concurrent scheduling algorithms on concurrent communication and a star and tree network topology are investigated using divisible load analysis [6, 7], since the star and tree topology is a good solution for master-worker style of parallel applications with independent divisible tasks.

Second, when both communication and computation on each processor are executed simultaneously, the computing capability experiences degradation due to the impact of communication on the same processor in networks and grids. It is meaningful to model communication interference on computation with estimating computing power more accurately in the presence of interference in networks and grids.

The third problem involves very large heterogeneous networks. Here, it will be increasingly common that multiple source nodes create and originate large amounts of data (workload) and any sink node can receive data from one of a number of source nodes. For high energy and nuclear physics experiments, large amounts of data originate from distant experiments. Such data requires a high computational power and network based computing platforms in these types of experiments. Minimizing the time to process workload originated from various sources presents a great challenge that could give rise to a range of new applications.

Finally, it has been also important to provide performance guarantees to deadline-constrained jobs originating from large scale experiments. To efficiently cope with these heavy workloads, divisible load theory (DLT) with real-time characteristics has been recently important in a heterogeneous grid/cluster systems, where the job with the earliest deadline is first scheduled. grid systems are inherently heterogeneous, since the different nodes in each site are connected to one another with different link speeds. A major trend is that cluster systems are also heterogeneous in the computing capability as well as in the link speed, the so-called heterogeneous cluster systems [8].

1.2 Motivation

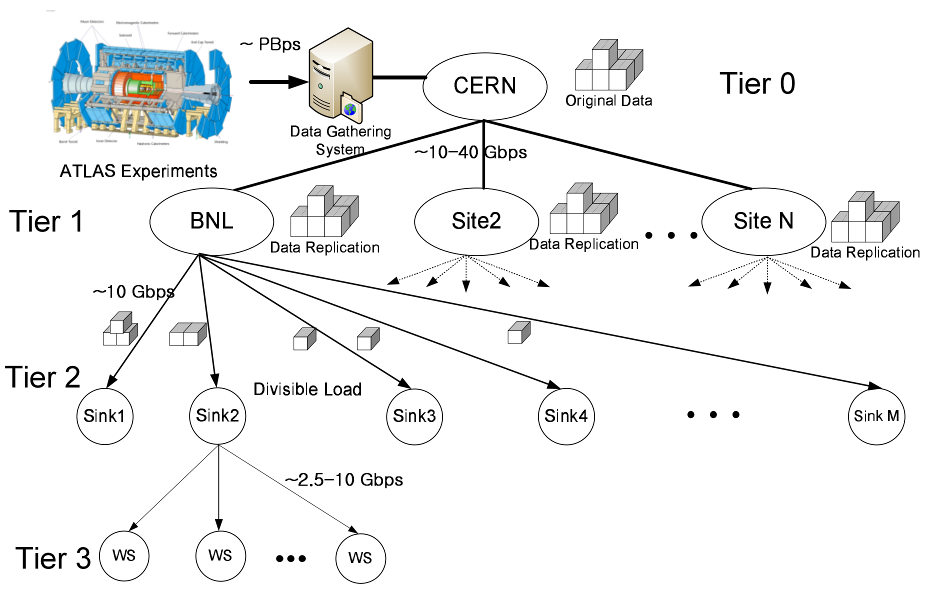


Figure 1-1: A multi-source heterogeneous Grid/cluster network configuration of the ATLAS project.

For high energy and nuclear physics experiments, large amounts of data originate from distant experiments. For example, a couple of universities in tier-2 sites should connected through a couple of WAN (wide area network)s to BNL (Brookhaven National Laboratory) in the ATLAS project [9, 10] as shown in Fig 1-1. In this configuration, large bulk data can be delivered from multiple sites in tier-1 to a site in tier-2. Also, each site can have heterogeneous cluster systems with different computing capability and there is a heterogeneous computing and link capability even within each cluster system. Specifically, the hardware of RACF (RHIC ATLAS Computing Facility) of STAR project consists of a combination of commodity-based processing servers, enterprise class UNIX servers and highly-specialized mass storage systems connected together by a high-speed network infrastructure. The RACF is an example of heterogeneous system with currently over 4000 processors and different link speeds. Such data require a high computational power and network based computing platforms. To satisfy the deadline of each job, sophisticated scheduling algorithms with respect to deadlines are needed. All of these recently emerging platforms require a sophisticated scheduling strategy to efficiently make use of distributed computers, high-speed networks and storage resources in terms of deadlines.

1.3 Contribution

For concurrent scheduling in parallel video processing, three contributions are made. First, a concurrent scheduling policy is significant for showing not only a more efficient scheduling method for parallel video encoding but also good scalability in the number of processors. Second, the extensive discussion here relating various interconnection topologies to this work should be of interest to other researchers. Third, the choice of scheduling policy has a greater impact on performance than whether or not the root node does computation for the network considered in this work. All in all divisible load modeling as it has been developed for parallel video processing by researchers including ourselves, has been shown to be a useful and cost effective tool for system performance prediction.

As for interference aware scheduling with sequential distribution in a tree topology, the following contributions are made: First, interference aware (IA) scheduling can be analytically modeled. Optimal load allocations, speedup and makespan are found in closed form equation. Second, interference aware scheduling produces a realistic modeling with a larger makespan and smaller speedup than modeling that does not take communication interference into account. Third, the results and policies with a tree topology here can be extended to other scheduling policies and interconnection topologies.

For multi-source scheduling, a specific network partitioning technique via the use of a genetic algorithm is proposed. The network is fragmented into several partitions matching the number of source nodes. Computation in each partition is independently performed on a single source and multiple sink nodes. That is, a complicated problem for multi-source scheduling is simplified into several single source scheduling problems after network partitioning. Through this algorithm we can obtain the effect of the concurrent communication with simultaneous distribution policy.

For deadline-constrained scheduling, real-time modeling from the perspective of DLT (divisible load theory) and application specific scheduling algorithms are proposed in a fully heterogeneous Gird/cluster systems. The minimum number of nodes obtained through a homogeneous model to satisfy the deadline of a job is applied to network partitioning. Through application specific scheduling it is possible to adapt the sequence of load distribution to the characteristics of a job. To enhance network performance guarantee for deadline constrained jobs, every job is scheduled with DLT (divisible load theory) and the EDF (earliest deadline first) policy where each job is sorted in the order of the earliest deadline to be evaluated if each job is schedulable within its deadline.

1.4 Outline

In this dissertation, four kinds of different scheduling methods, concurrent scheduling and interference aware scheduling, multi-source scheduling, and deadline-constrained scheduling algorithms are proposed for parallel system and fully heterogeneous grid networks. These algorithms deal with concurrent communication in parallel video processing, communication interference on processing, workloads originating from multiple sources, and deadlines in real-time systems, respectively. The primary network is a fully heterogeneous tree network which is commonly used in parallel/cluster systems and grid networks.

In chapter 2, a simultaneous distribution scheduling method for parallel video processing based on multi-port communication is proposed. Here we discuss an efficient scheduling mechanism, SD (Simultaneous Distribution), for parallel video processing which distributes raw video loads and collects encoded video results concurrently among the root (control) processor and each child worker processor in a star topology with a multi-port interconnect. We consider two cases: one is that load is assigned to the root processor (SD-COMP) and the other is that no load is assigned to the root processor (SD-NO). For the two cases, we obtain closed-form solutions for the total video processing time, and then compare these results in terms of the performance under the optimal number of processors which is proposed for previous scheduling algorithms, such as PI and PR [1, 11]. Both of the two cases using our strategies show much better performance in video processing and several times less finish time for the parameters we use than those under the optimal number of processors of previous methods, such as PI and PR. From the practical point of view, feasible hardware interconnect networks, such as fat-tree or a multi-port tree network, are investigated and proposed for concurrent scheduling methods in parallel video processing.

In chapter 3, we consider communication interference on computation and propose Interference aware (IA) scheduling method to realistically model sequential distribution in a tree network. It is a very common situation in networks and grids that both communication and computation on each processor are executed simultaneously. When the impact of communication in sending or receiving load is considered, the traditional divisible load sequential distribution model should be modified. In this paper we analytically develop an optimal scheduling policy in the presence of interference of communication on computation for the sequential distribution and simultaneous start scheduling method which is one of traditional DLT (divisible load theory) models [12,13]. This particular load distribution policy is chosen for illustrative purposes-certainly other scheduling policies could be modeled in an interference aware context. In addition, the interference aware scheduling method considering concurrent communication (IA-COMP) is proposed and applied to parallel video processing. The IA-COMP method shows a little bit slow finish time due to the communication interference than the SD-COMP method, but the IA-COMP is found to be more realistic and accurate modeling.

In chapter 4, a multi-source scheduling scheme through network partitioning is proposed. All sink nodes are involved in only one partition associated with a single source node. In partitioning the network, the sum of computing speeds of the sink nodes in each partition should be similar one another for the initial network partition. Each partition is evaluated by running an optimal single-source sequential load distribution scheduling method whenever network partitioning is repeated. After every evaluation, new partitions could be constructed by transferring sink nodes from one partition to another or rearranging the sequence of sink nodes receiving load from the source node within a partition. We iterate this partitioning and evaluation via a genetic algorithm until a globally near-optimal solution is approached or obtained.

In chapter 5, divisible load theory (DLT) has been extended with real-time characteristics, where the job with the earliest deadline is the first scheduled. To satisfy the deadline of workloads, computing and communication capability should be guaranteed and each workload should be estimated through a schedulability test before submission. In this chapter, deadline-constrained network partitioning scheduling is proposed for large scale heterogeneous grid/cluster systems. The minimum number of nodes obtained from a homogeneous model is used in testing the schedulability of a job in the original heterogeneous system. The entire network is fragmented into small partitions with the minimum number of processors for each job with deadline constraint, where the minimum number of children nodes are selected to be just enough to satisfy the deadline of each job.

Chapter 2

Concurrent Scheduling

In parallel video processing, various scheduling algorithms have been presented such as PI (parallel interlaced) and PR (parallel recursive) which can assign video frames to multiple processors. For these two algorithms other researchers [1, 11] found both the maximum processing throughput and I/O utilization, and the optimal number of processors for each of algorithms under a bus architecture, using divisible load analysis [6, 7]. However the scheduling algorithms have inherent limitations of a bus architecture on communication in terms of throughput and the optimal number of processors.

A scheduling method considering result collection as well as load distribution overheads was first proposed by Barlas [14] in modeling the divisible load like video processing and database query processing. Those applications are based on architecture which shares a single communication channel and which is modeled on a tree topology which consists of the single root node and several children nodes. In [5], divisible load like video frames are considered with respect to software functionality to minimize the processing time of the video encoding on a bus architecture. Each video frame is divided into 16 x 16 blocks and each block is distributed to children processors. Since software functionality like motion estimation demanding much time is performed for each divided block on each child processor, this load partition scheme obtains a good performance in parallel video encoding. Results and problems in scheduling divisible load on a star and tree network (including a bus architecture) were covered in [15]. The authors show that the star and tree topology is a good solution for master-worker style of parallel applications with independent divisible tasks.

Here, we propose an efficient scheduling mechanism, SD (Simultaneous Distribution), for parallel video processing which distributes raw video loads and collects encoded video results concurrently among the root (control) processor and each child worker processor in a star topology with a multi-port interconnect. Note that simultaneous distribution was proposed by Piriyakumar and Murthy [16] and analyzed by Hung and Robertazzi [17]. We consider two cases: one is that load is assigned to the root processor and the other is that no load is assigned to the root processor. For the two cases, we obtain closed-form solutions for the total video processing time, and then compare these results in terms of the performance under the optimal number of processors which is proposed for previous scheduling algorithms, such as PI and PR [1, 11]. Both of the two cases using our strategies show much better performance in video processing and several times less finish time for the parameters we use than those under the optimal number of processors of previous methods, such as PI and PR. In terms of the number of scalable processors, our proposed method, SD, reaches up more than twice the optimal number of processors of PI or PR .

Of practical interest is that we propose a multi-port star topology among the root (control) processor and children worker processors. This means that the control processor has ports to each of the children processors for I/O communication. One of the reasons to select the multi-port star topology is that there is only communication between the root processor and each of children processors without communication among children processors. The other aspect is that the star topology is cost effective model for parallel video processing and relatively simple to implement compared with other complex architectures, such as 2D meshes, or Hypercubes.

We know that when the number of processors is small, the factors that affect the total processing time are the method to distribute and collect load as well as the root processor participation in computation. As the number of processors increases, all of the simultaneous scheduling methods (SD) show better performance than each of the sequential scheduling methods, such as PI or PR, because all of the SD methods have good scalability. However when the number of processors is 30, the performance improvement of the SD-COMP method (SD with computation) is 6 times as much as that of the sequential distribution method, PI. As for the SD-NO (SD with NO computation) method, the improvement of the SD-COMP with respect to SD-NO is just 1.3 times better. This is because the most critical part in efficient load distribution is how to distribute and collect load rather than whether or not the root processor involves load computation, when the number of processors is large enough to process the whole load.

2.1 Concurrent Communication In Parallel Video Processing

2.1.1 Interconnect Topology

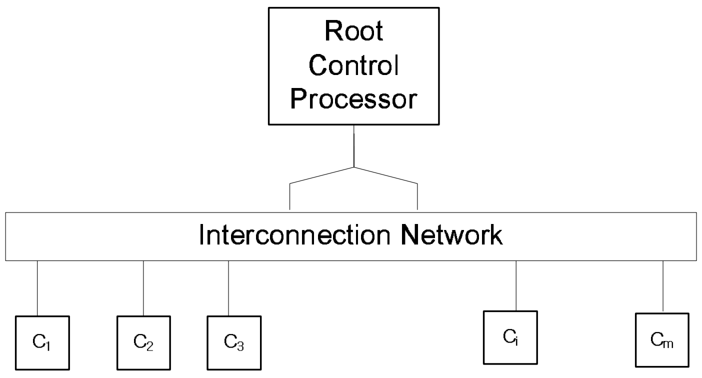


Figure 2-1: Block diagram for multi-port interconnection network.

In this section, a one-to-many interconnect is considered, which consists of one root (control) processor with multiple ports and m children processors. The root (control) processor distributes raw video data (load) and collects the encoded video data (results) to/from each child processor simultaneously via multiple ports. While the children processors encode the video, the root processor waits for the encoded video data from each child processor.

From the perspective of hardware implementation, the above interconnect can be logically modeled on various topologies such as a star, fat tree, hypercube, and mesh/torus topology. For a hypercube topology, the simultaneous use of links was proposed to obtain faster communication and it was found that there is no need to use all the processors to obtain an optimal solution. In this hypercube topology each processor needs multiple ports, for example, 8 ports in the *Intel iPSC*/860, to concurrently communicate each other [18]. For a 3D-mesh topology, we can use multiple links to simultaneously communicate, but its ports are constrained 1 to 6 in commercially available computers, such as the *Cray X3D*. Similarly in the *Cray XT3* computer using a 3D-torus topology, each processor has 6 ports [19, 20].

The above hypercube and 3D-mesh, 3D-torus interconnects are not appropriate for parallel video processing using our methods as all of the processors in the networks have multiple ports. Our star topology requires each child processor to have only one port. The root processor in our scheduling methods is the only processor to have multiple ports. Therefore we take into account more suitable solutions for our scheduling method based on a master-slave structure and analyze the complexity, feasibility, and cost-effectiveness in terms of the implementation point of view.

Topology	# of Node	# of Links	Degree	Network Diameter	Model
Hypercube	$N = 2^n$	$n\frac{N}{2}$	n	n	*Intel iPSC*
3D-Mesh	$N = p^{(3)}$	$3p^2(p-1)$	4 6	3(p-1)	*Cray X3D*
3D-Torus	$N = p^{(3)}$	$3p^{(3)}$	6	$3\lfloor\frac{p}{3}\rfloor$	*Cray XT3*
Binary Fat-tree	$N = 2^h - 1$	N-1	3	$2(\log_2 N - 1)$	*CM5*
Star	N	N-1	N-1	2	*Cray XD1*
Multi-port Memory	N	N-1	N-1	2	*IBM RS6000*
Optical BUS	N	N	1	1	*Cray T90*

Table 2.1: The Properties of topologies. $'n'$: a dimension of Hypercube, $'p'$: the number of nodes along one edge of Mesh and Torus, $'h'$: the height of the binary tree. (*K.Hwang et al.* [2], *Duato et al.* [3], *http : //www.netlib.org/benchmark/top500* [4]).

In previous work, we proposed a method which has output buffers in the root processor to be used for output ports as a way to implement simultaneous communication [17]. This can be implemented via multi-port memory as long as each child processor has a different memory partition in the root processor.

In a star topology, the root processor uses multiple ports with a direct interconnect to simultaneously communicate with children processor. For example, there are switching devices which support 12 communication ports per chassis and can be expanded to several hundreds of processors in commercially manufactured system like $Cray\ XD1$ [20]. In this star topology the root node only has multiple ports and so it can be suitable for parallel video processing applications in terms of cost-effectiveness. This is because resources like the frame buffer in which raw and encoded image data are stored, are placed at the root node and only accessed through the root processor by all children processors in a star topology. The root node sends and receives data to or from a port of each child and then simply extracts or stores the data from the port to the frame buffer. Extending the star topology means increasing the fan-out of the root node. This makes the growth complexity one, which is better than most of other topologies such as 3D-Mesh, 3D-Torus, and Hypercube. The root node has to be modified in order to cope with an extra node, while all the other nodes can remain unchanged. A disadvantage of this topology lies in the fact that the root processor can become a communication bottleneck. However, since there is little communication among children processors in our proposed scheduling methods, this communication bottleneck does not significantly affect the overall performance.

In a fat tree topology, processors are located at the terminal nodes and switches are at the internal nodes. Transmission bandwidth in a fat-tree is increased by adding more links as nodes moves up the tree close to the root. For example, to alleviate the bottleneck of nodes close to the root node the commercial model $CM5$ used a four-way interconnect for each node to have four children nodes. If we only consider the performance, a fat-tree topology can be a good alternative to our methods. However, with respect to cost-effectiveness this topology requires more switches and links to connect processors than those of a star topology.

On the other hand, we can consider a optical bus topology to satisfy the simultaneous communication on the bus architecture in implementing our parallel video processing application. For example, the *Jitney* Optical Bus with 20 channels ($500Mb/s/ch$) has been designed for high speed parallel computing and successfully demonstrated in $IBM\ AS/400$ and $RS6000$ power parallel systems test-beds [21]. Meanwhile, as system frequencies move into from the MHz range to the GHz range, shared buses are generally migrated into point-to-point switches. Implementing switches within a chip limits the number of ports per chip to 6 or 8 ports. Currently, optical interconnect is generally used in packet-switched point-to-point network topologies, such as 3D Torus and fat-tree. Nevertheless, the models with reconfigurable optical buses are likely to become feasible architectures in the near future [22].

In summary, although there are tradeoffs, star and tree architectures are most feasible for our proposed scheduling methods in video processing from the cost-effectiveness point of view.

2.1.2 Scheduling Scenario

We have two scenarios for simultaneous scheduling. The first scenario is for the root processor to only distribute and collect load without computation. This is because we try to compare its performance with the sequential method in previous papers [3, 4]. Here we assume that our multi-port star network is homogeneous, which means all of the children processors are identical in terms of the computing speed. In addition, the communication speed between the root processor and each child processor is also identical. The other case is for the root processor to do both communication, such as load distribution and result collection, and some of the computation (video encoding). Here we assume that all of the children processors are homogeneous in terms of computing speed and communication speed as in the previous scenario, but the root processor speed can be different from the children's speed.

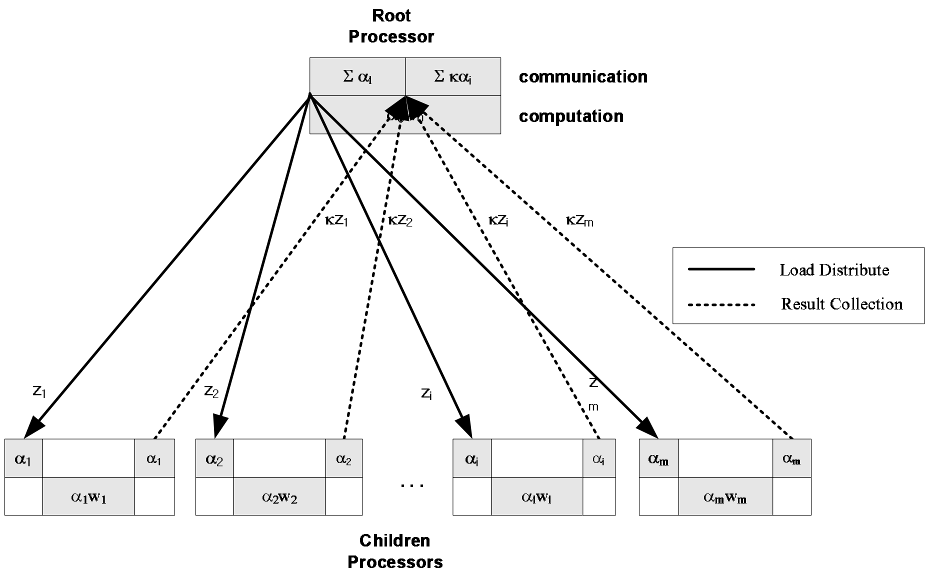


Figure 2-2: The simultaneous load distribution in a tree network.

In Fig. 2-2, the value of $'k'$ is defined as the ratio of the amount of result (an encoded video) obtained from each child processor to the amount of load sent (an original raw video). That is, the fraction of load may or may not be assigned to the root processor. If it is assigned, the root processor not only distributes load and collects results to/from each child processor, but also joins computation itself. Otherwise, the root processor just distributes and collects load. Here $'k'$ is the ratio of the result received to the original load sent.

$$k = \frac{result\_received}{load\_sent}$$

We have the three cases as follows:

- $k = 1$, if the amount of load sent is same as the amount of result received.
- $k < 1$, if the amount of load sent is greater than the amount of result received.

This case is typical in digital video processing due to compression.

- $k > 1$, if the amount of load sent is less than the amount of result received.

2.1.3 Notation

The variables we will use in the following are based on work in the papers [6, 7, 11].

α_i : The load fraction assigned to the ith link-processor pair (where $i = 0, 1, 2, \ldots, m$).

w_i : The inverse computing speed at the ith processor (where $i = 0, 1, 2, \ldots, m$).

z_i : The inverse communication speed on the ith link (where $i = 0, 1, 2, \ldots, m$).

T_{cp} : Computing intensity constant.

T_{cm} : Communication intensity constant.

$T_{f,m}$: The finish time. Time at which each processor completes computation.

Then $\alpha_i w_i T_{cp}$ is the time to process the fraction i of the entire load on the ith processor. Note that the units of $\alpha_i w_i T_{cp}$ are [load] x [time/load] x [dimensionless quantity] = time. Likewise, $\alpha_i z_i T_{cm}$ is the time to transmit the fraction i of the entire load over the ith link. Our goal is to propose more efficient scheduling methods and analyze the solution in parallel video processing through concurrent communication.

2.2 Simultaneous Distribution Scheduling (SD)

We assume that the root processor has a faster computing speed than that of the children processors, while all of the children are identical in terms of computing speed and link speed. We consider the case of a homogenous processor network, which means all children processors except the root processor are identical. ; the inverse processor speed is $w_1 = w_2 = \ldots = w_m = w$ and the inverse network speed is $z_1 = z_2 = \ldots = z_m = z$.

2.2.1 No computation on the root processor (SD-NO)

In this strategy the root processor does not join computation by itself, but just distributes load and collects results to/from the children processors. The timing diagram for concurrent scheduling is shown in Fig. 2-3, which distributes load simultaneously to the children processors. Here the root processor does not execute computation in itself, but just distributes and collects load. In all of the scheduling policies to be considered we force all activities to terminate at the same time instant, as otherwise load could be redistributed for a better solution [12, 13]. From the timing diagram for SD-NO, as shown in Fig. 2-3, the equations for SD-NO scheduling are obtained as follows.

$$\alpha_1 z T_{cm} + \alpha_1 w T_{cp} + \alpha_1 k z T_{cm} = \alpha_2 z T_{cm} + \alpha_2 w T_{cp} + \alpha_2 k z T_{cm} \tag{2.1}$$

$$\alpha_1 = \frac{(z + kz)T_{cm} + wT_{cp}}{(z + kz)T_{cm} + wT_{cp}} \alpha_2 = \alpha_2 \tag{2.2}$$

From equation (2.2), we deduce as follows:

$$\alpha_1 = \alpha_2 = \alpha_3 = \ldots = \alpha_m \tag{2.3}$$

The normalization equation that all of the load fractions is summed up 1 is

$$\sum_{i=1}^{m} \alpha_i = 1 \tag{2.4}$$

From equation (2.3) and (2.4), we obtain

$$\alpha_1 \times m = 1, \qquad \alpha_1 = \frac{1}{m} \tag{2.5}$$

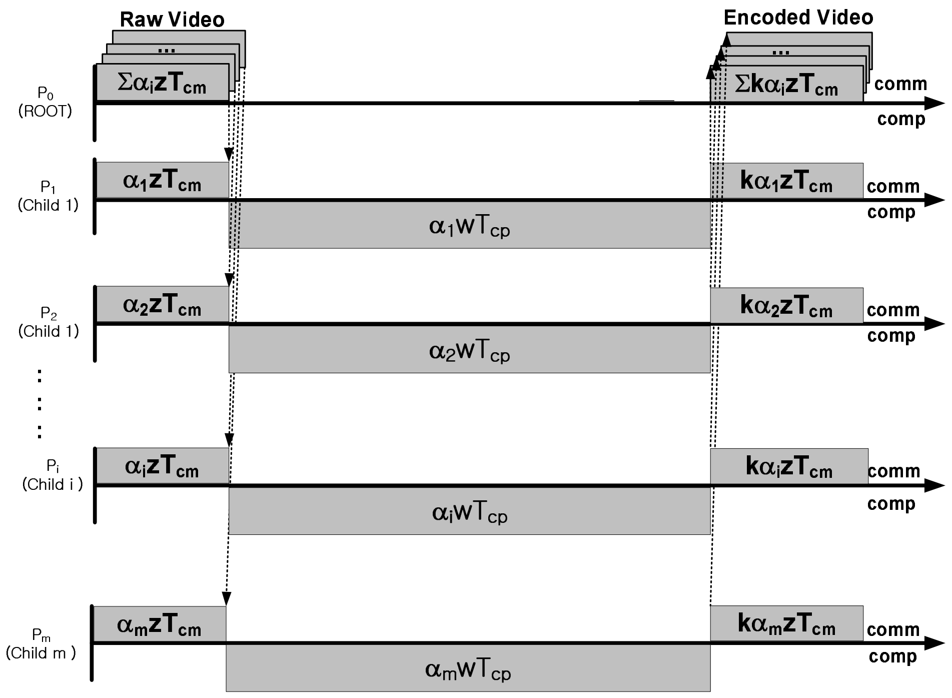


Figure 2-3: The timing diagram for the SD-NO method.

$$\alpha_m \times m = 1, \qquad \alpha_m = \frac{1}{m} \tag{2.6}$$

The total processing time for the entire load, $T_{f,m}$ is achieved as

$$T_{f,m} = \alpha_1 zT_{cm} + \alpha_1 wT_{cp} + \alpha_1 kzT_{cm} = \alpha_1(1 + \sigma + k\sigma)wT_{cp} \tag{2.7}$$

$$where \quad \sigma = \frac{zT_{cm}}{wT_{cp}}$$

From the above equation (2.5), the total processing time, $T_{f,m}$ can be rewritten as follows:

$$T_{f,m} = \frac{(1 + \sigma + k\sigma)}{m} wT_{cp} \tag{2.8}$$

Our finding is that the total processing time decreases in inverse proportion to the number of children processors.

2.2.2 Computation on the root processor (SD-COMP)

In the case load is assigned to the root processor and some computation is done on the root processor itself. As for the root processor, the processor speed is greater than those of children processors, which means the inverse value of the root processor, w_0, is less than w.

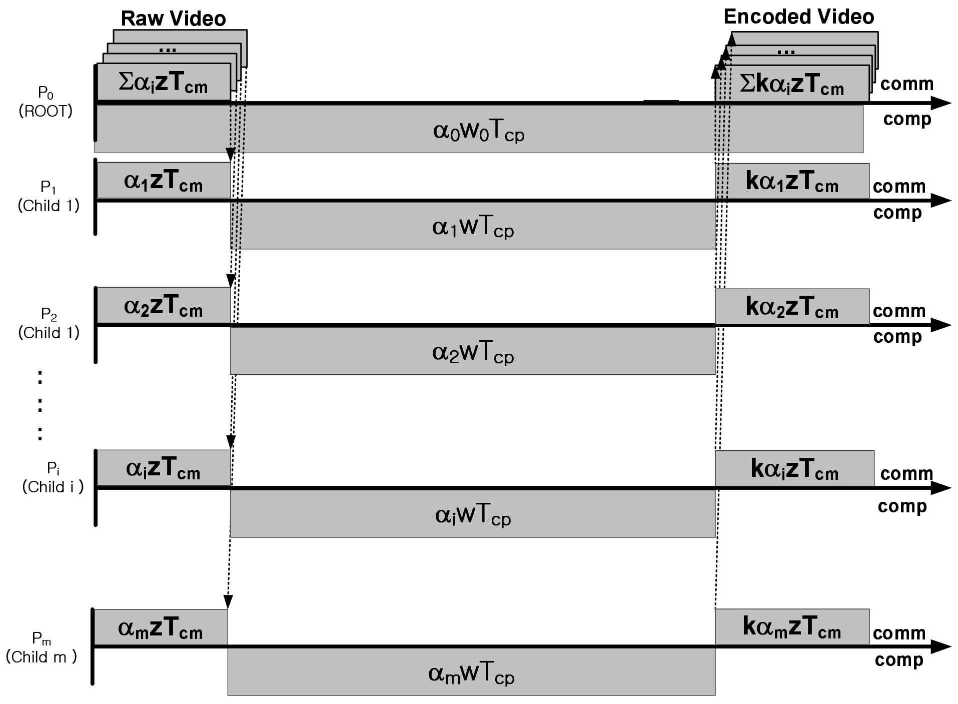


Figure 2-4: The timing diagram for the SD-COMP method.

The SD-COMP scheduling method is illustrated in Fig. 2-4, where load is simultaneously distributed to the children processors and the root (control) processor computes load assigned to itself as well as distributes and collects load. From the timing diagram in Fig. 2-4, the equations for the SD-COMP method, in which the root processor has load assigned, are obtained as follows:

$$\alpha_0 w_0 T_{cp} = \alpha_1 z_1 T_{cm} + \alpha_1 w_1 T_{cp} + \alpha_1 k z_1 T_{cm} \tag{2.9}$$

$$\alpha_1 z_1 T_{cm} + \alpha_1 w_1 T_{cp} + \alpha_1 k z_1 T_{cm} = \alpha_2 z_2 T_{cm} + \alpha_2 w_2 T_{cp} + \alpha_2 k z_2 T_{cm} \tag{2.10}$$

$$\ldots$$

$$\begin{aligned} &\alpha_{m-1} z_{m-1} T_{cm} + \alpha_{m-1} w_{m-1} T_{cp} + \alpha_{m-1} k z_{m-1} T_{cm} \\ &= \alpha_m z_m T_{cm} + \alpha_m w_m T_{cp} + \alpha_m k z_m T_{cm} \end{aligned} \tag{2.11}$$

The normalization equation is different from the previous one in the SD-NO method, in that the load fraction to the root processor, α_0 is added as follows

$$\alpha_0 + \alpha_1 + \alpha_2 + \ldots + \alpha_m = 1 \tag{2.12}$$

From equation (2.9),

$$\alpha_0 = \frac{[(z_1 + k z_1) T_{cm} + w_1 T_{cp}]}{w_0 T_{cp}} \alpha_1 = \frac{1}{\gamma_1} \alpha_1 \tag{2.13}$$

$$where \qquad \gamma_1 = \frac{w_0 T_{cp}}{[(z_1 + k z_1) T_{cm} + w_1 T_{cp}]}$$

From equation (2.11),

$$\alpha_i = \frac{[w_{i-1}T_{cp} + (z_{i-1} + kz_{i-1})T_{cm}]}{w_iT_{cp} + (z_i + kz_i)T_{cm}}\alpha_{i-1} = q_i\alpha_{i-1} \tag{2.14}$$

$$where \qquad q_i = \frac{[w_{i-1}T_{cp} + (z_{i-1} + kz_{i-1})T_{cm}]}{w_iT_{cp} + (z_i + kz_i)T_{cm}}, \qquad i = 2, 3, \dots\ m$$

Equation (2.14) can be represented as

$$\alpha_i = q_i\alpha_{i-1} = (\prod_{l=2}^{i} q_l)\alpha_1 \tag{2.15}$$

$$i = 2, 3, \dots\ m$$

From equations (2.9),(2.11), the normalization equation (2.12) becomes

$$\frac{1}{\gamma_1}\alpha_1 + \alpha_1 + \alpha_2 + \dots + \alpha_m = \frac{1}{\gamma_1}\alpha_1 + \alpha_1 + \sum_{i=2}^{m} \alpha_i = 1 \tag{2.16}$$

$$[\frac{1}{\gamma_1} + 1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)]\alpha_1 = 1 \tag{2.17}$$

$$\alpha_1 = \frac{1}{[\frac{1}{\gamma_1} + 1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)]} \tag{2.18}$$

From the timing diagram, Fig. 2-4, we can obtain the finish time with m+1 processors,

$T_{f,m}$, as follows:

$$T_{f,m} = \alpha_0 w_0 T_{cp} = \frac{1}{\gamma_1} \alpha_1 w_0 T_{cp} \tag{2.19}$$

While the finish time with only one processor, $T_{f,0}$, is

$$T_{f,0} = \alpha_0 w_0 T_{cp} = w_0 T_{cp} \tag{2.20}$$

The speed-up, which is the ratio of job finish time of one processor to that on $m + 1$ processors, can be obtained as:

$$Speedup = \frac{T_{f,0}}{T_{f,m}} = \gamma_1 \times \frac{1}{\alpha_1} = 1 + \gamma_1 [1 + \sum_{i=2}^{m} (\prod_{l=2}^{i} q_l)] \tag{2.21}$$

Since $\prod_{l=2}^{i} q_l$ can be simplified as $\frac{[w_1 T_{cp} + (z_1 + kz_1) T_{cm}]}{w_i T_{cp} + (z_i + kz_i) T_{cm}}$, the speed-up, *Speedup* in equation (2.21)and the finish time, $T_{f,m}$ in equation (2.19)can be derived as follows:

$$T_{f,m} = \frac{1}{\gamma_1} \alpha_1 w_0 T_{cp} = \frac{w_0 T_{cp}}{1 + \gamma_1 \left(1 + \sum_{i=2}^{m} \frac{[w_1 T_{cp} + (z_1 + kz_1) T_{cm}]}{w_i T_{cp} + (z_i + kz_i) T_{cm}}\right)} \tag{2.22}$$

$$Speedup = 1 + \gamma_1 \left[1 + \sum_{i=2}^{m} \frac{[w_1 T_{cp} + (z_1 + kz_1) T_{cm}]}{w_i T_{cp} + (z_i + kz_i) T_{cm}}\right] \tag{2.23}$$

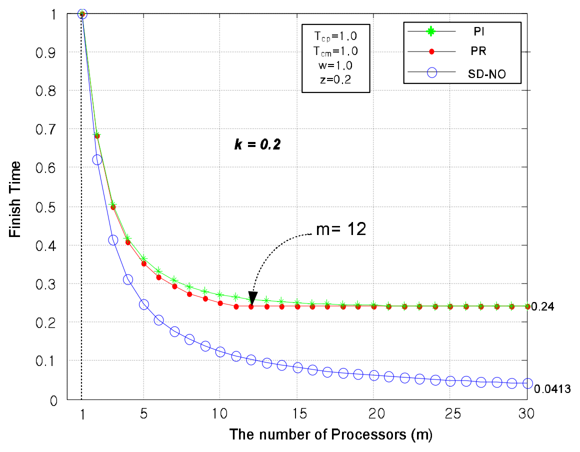
For a special case, a homogeneous network, all of children processors have the same processing speed and all of the links have the same transmission speed, the finish time, $T_{f,m}$ is

$$\begin{aligned} T_{f,m} &= \frac{w_0 T_{cp}}{1+\gamma_1[1+(m-1)]} = \frac{w_0 T_{cp}}{1+\gamma_1 \times m} \\ &= \frac{w_0 T_{cp}}{1+m\times\left[\frac{(1+K)zT_{cm}+wT_{cp}}{w_0 T_{cp}}\right]} \\ &= \frac{w_0 T_{cp}}{1+m\times[(1+K)\sigma^* + q^*]} \end{aligned} \tag{2.24}$$

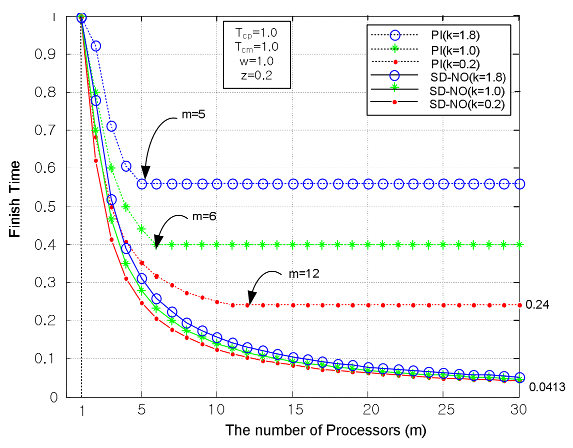
where γ_1 is from (2.13), $\sigma^* = \frac{zT_{cm}}{w_0 T_{cp}}$, and $q^* = \frac{w}{w_0}$. From (2.21), speed up for a homogeneous network (all children processing speed and all link speed are identical) is obtained as follows:

$$\begin{aligned} Speedup = \frac{T_{f,0}}{T_{f,m}} &= 1+\gamma_1[1+\sum_{i=2}^{m} 1] \\ &= 1+\gamma_1 \times m \end{aligned} \tag{2.25}$$

Since the inverse computing speed, w and the inverse communication speed, z is identical among all processors and links in homogeneous network respectively, the value of q_i from (2.14) not only becomes one, but the value of $\prod_{l=2}^{i} q_l$ is also one. It can be seen that the value of speedup is linearly related to the number of processors in a simultaneous distribution and collection method.



(a) $k = 0.2$



(b) $k = 0.2,\ 1.0,\ 1.8$

Figure 2-5: The finish time versus the number of processors for SD-NO, PI and PR load scheduling methods.

2.3 Performance Analysis And Comparison

2.3.1 Speedup of the SD-NO method

In this section, for the SD-NO (Simultaneous Distribution with NO computation) scheduling method, we assume that the root processor is identical to each child processor in terms of computing speed. The root processor does not have load assigned to itself, but just distributes and collects load to/from children processors, We consider the same parameters as those of PI and PR in papers [1,11]. The inverse computing speed of the processor, w, is 1.0, and the inverse communication speed, z, is 0.2. Both T_{cp} and T_{cm} are 1.0. Three values of the ratio, k, are considered: 0.2, 1.0, and 1.8.

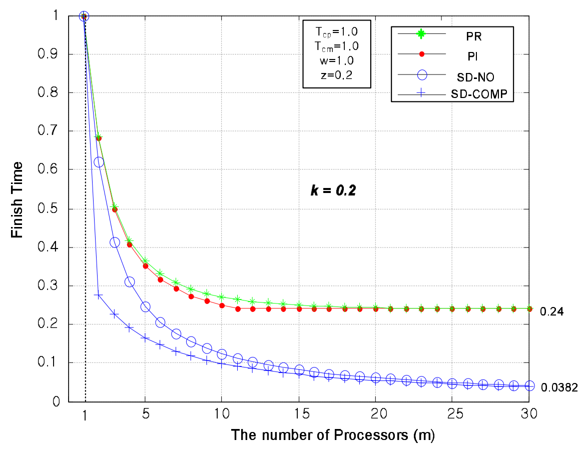
In Fig. 2-5 (a), our load scheduling mechanism, SD-NO shows a much better performance than PI and PR. When the number of processors is 12, which is the optimal number of the processors in PI, the SD-NO method shows more than 2 times less processing time as PI and PR. Especially when we consider more processors added in the network, for example, 30, the difference is much larger, which is above 6 times for PI. This means our mechanism, SD-NO, is more scalable and cost effective in terms of the computing speed. When the number of processors increases from 12 to 30, the performance of the system increases almost 6 times, while the number of processors only increases 2.5 times.

In Fig. 2-5 (b), we see that for all three cases of the ratio, k, where $k < 1, k = 1, k > 1$, our mechanism shows a much better performance than that of PI. In terms of the optimal number of processors, SD-NO shows almost 2 times better performance than that of PI for three $'k'$ values. When we consider processor scalability, for a number of processors of 30, SD-NO achieves much better performance than that of PI. That is more than 10 times, 8 times, and 6 times improvement for each of $k > 1$, $k = 1$, $k < 1$.

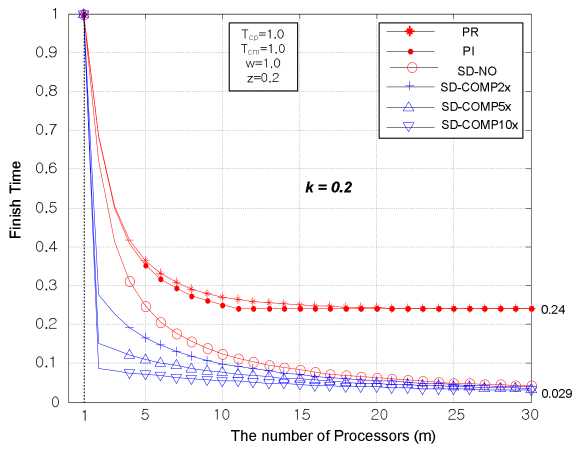
2.3.2 Speedup of the SD-COMP method

In this section, for the SD-COMP (Simultaneous Distribution with Computation) scheduling method, we assume that the root processor is different from the children in terms of computing speed and has load to compute itself. So the root processor not only distributes and collects load to/from children processors, but also computes load. The ratio of load received to load sent, k, is chosen as 0.2, since we suppose the case k is less than 1, as is usually the case for compressed results.

In Fig. 2-6 (a), we assume that the computing speed of the root processor for



(a) The root processor is twice as fast as a child processor



(b) The root processor is twice, 5 times, and 10 times as fast as a child processor.

Figure 2-6: The finish time versus the number of processors for SD-COMP, SD-NO, PI, and PR load scheduling methods on a homogeneous network.

SD-COMP is twice as much as that of each child processor. That is the inverse computing speed of the root processor, w_0, is half of that of each child, $'w'$. We see that SD-COMP method is continuously faster for SD-NO method, and much faster, for example more than 6 times, for PI and PR method up to the number of processors, 30. In terms of processor scalability, SD-COMP has a more improved result. When the number of processors increases from 12 to 30, the performance of SD-COMP goes up 2.24 times to 6 times as fast as respectively that of PI. However, SD-COMP and SD-NO method shows similar performance and good scalability.

From Fig. 2-6 (b), we consider three cases of computing speed of the root processor for SD-COMP.Those are twice, 5 times, and 10 times as fast as that of each child processor. When the number of processors is small, for example 2 to 5, the performance of the SD-COMP method is much better than the SD-NO, PI, and PR methods, because the root processor of the SD-COMP method participates in computation itself, involving around 20% to half of the whole load. While the number of processors increases to 12, all of the SD methods show 2.4 times, 3 times, 4 times, 5 times improvement in the processing time irrespective of load assigned to the root processor. As the number of processors increases up to 30, all of the SD methods show still better performance than the PI and PR methods, because all of the SD methods have good scalability in the number of processors. However when the number of processors is 30, the performance improvement of the SD-COMP is small, just 1.3 times, against the SD-NO method as compared to 6 to 8 times against PI and PR. One point to note is that when the number of processors is small, it is the method to distribute load as well as the root processor speed that is important to total processing time. The other point is that when the number of processors increases enough, the most critical part is the method to distribute and collect load simultaneously or sequentially rather than whether load is assigned to the root processor.

This work is meaningful for showing not only a more efficient scheduling method for parallel video encoding, but also good scalability in the number of processors.

2.4 Conclusion

This work leads to the following conclusions:

• This work is meaningful for showing not only a more efficient scheduling method for parallel video encoding but also good scalability in the number of processors.

• This work is novel in proposing a method of modeling interference aware computation that leads to more realistic results.

• The extensive discussion here relating various interconnection topologies to this work should be of interest to other researchers.

• We find that the choice of scheduling policy has a greater impact on performance than whether or not the root node does computation for the network considered in this work.

• Finally many factors influence performance results obtained through mathematical (divisible) modeling including scheduling policy, interconnection topology, memory hierarchy, fixed communication delays, and the potential use of front-end processors. Most papers, like this one, consider a small number of these factors for reasons of space, tractability and novelty but the most accurate modeling would take most if not all of these factors into account.

All in all divisible load modeling as it has been developed for parallel video processing by researchers including ourselves, has been shown to be a useful and cost effective tool for system performance prediction.

Chapter 3

Interference Aware Scheduling

In a situation in networks and grids that both communication and computation on each processor are executed simultaneously, the computing speed experiences degradation due to the impact of communication on the same processor. Here we propose a scheduling method considering communication interference, so called , interference aware (IA) scheduling, and develop analytically an optimal scheduling policy in the presence of interference of communication on computation for the sequential distribution. This particular load distribution policy is chosen for illustrative purposes-certainly other scheduling policies could be modeled in an interference aware context.

When load is distributed to children processors, the computing speeds of the root and children processors are degraded by communication (sending and receiving load). Only during certain phases of load distribution and processing is communication active. It is only during these times that computation is affected by communication. We define the affected inverse computing speed, $w_i^{'}$, and the unaffected inverse computing speed, $w_i^{''}$ separately. The load fraction for each processor, α_i, is also divided into two portions, $\alpha_i^{'}$ for the affected load and $\alpha_i^{''}$ for the unaffected load. From the relationship among the processors in our model, we find $3m + 3$ unknown variables and $3m + 3$ equations where $m + 1$ is the number of processors. We can reduce the number of equations from $3m+3$ to $m+1$ by using the equivalent processor concept and techniques, which means that both the affected and unaffected processing capability of each processor can be merged into a single processor which has equivalent processing capability.

3.1 Sequential Distribution Model and definition

In this paper sequential distribution and simultaneous start scheduling is investigated under the interference of communication on computation in a heterogeneous single level tree (star) topology.

3.1.1 Sequential distribution model for a heterogeneous tree

When the impact of communication in sending or receiving load is considered, the traditional sequential distribution model is slightly changed as in Fig. 3-1. The computing speed of each processor is decreased only when communication and computation overlap. This means that the original inverse computing speed, w_i, is changed into $w_i^{'}$ in case computation overlaps sending or receiving load. The load fraction to each processor, α_i, is divided into $\alpha_i^{'}$ for the affected computation and $\alpha_i^{''}$ for unaffected computation.

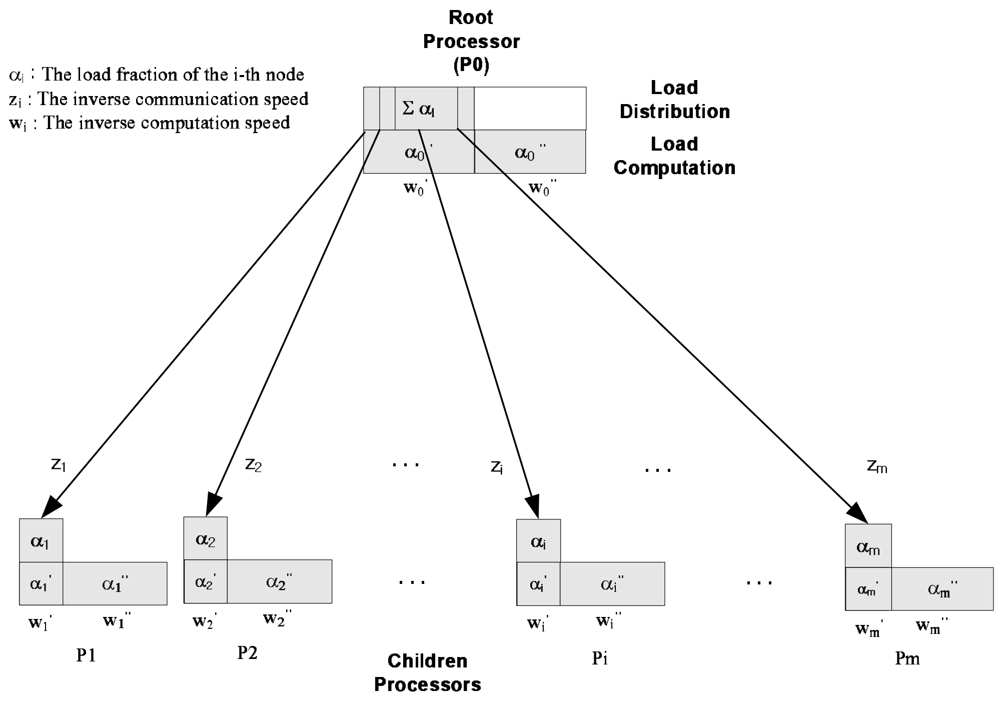


Figure 3-1: Interference aware sequential load distribution in a tree topology.

3.1.2 Notation

α_i : The load fraction assigned to the ith processor (where $i = 0, 1, 2, \ldots, m$).

$\alpha_i^{'}$: The load fraction assigned to the ith processor with communication (where $i = 0, 1, 2, \ldots, m$).

$\alpha_i^{''}$: The load fraction assigned to the ith processor without communication (where $i = 0, 1, 2, \ldots, m$).

w_i : The inverse computing speed at the ith processor (where $i = 0, 1, 2, \ldots, m$).

$w_i^{'}$: The inverse computing speed at the ith processor when sending or receiving load (where $i = 0, 1, 2, \ldots, m$).

$w_i^{''}$: The inverse computing speed at the ith processor which is not affected by the communication (where $i = 0, 1, 2, \ldots, m$).

w_i^{eq} : The equivalent inverse computing speed at the equivalent node, $node^{eq}_{<i>}$, collapsed from a single level tree rooted at $node_{<i>}$ (where $i = 0, 1, 2, \ldots, m$).

z_i : The inverse communication speed on the ith link (where $i = 0, 1, 2, \ldots, m$).

T_{cp} : Computing intensity constant.

T_{cm} : Communication intensity constant.

$T_{f,m}$: The finish time (makespan). Time at which the root processor and m children processors complete computation.

$T_{f,0}$: The finish time. Time at which computation ends for a single root processor.

3.2 Interference aware scheduling (IA)

3.2.1 Timing Diagram for Sequential Distribution

The mechanism of interference aware scheduling via sequential load distribution and simultaneous start is presented in Fig. 3-2. From this timing diagram, we obtain the relationships among the processors. The following equations is used for this sequentially distributed interference aware model. Since there are $3m+3$ unknown variables and $3m + 3$ equations, we can deduce the corresponding closed-form solution. From the seven basic equations we can define all of the relationships between processors.

First, the normalization equation consists of the sum of all load fractions assigned

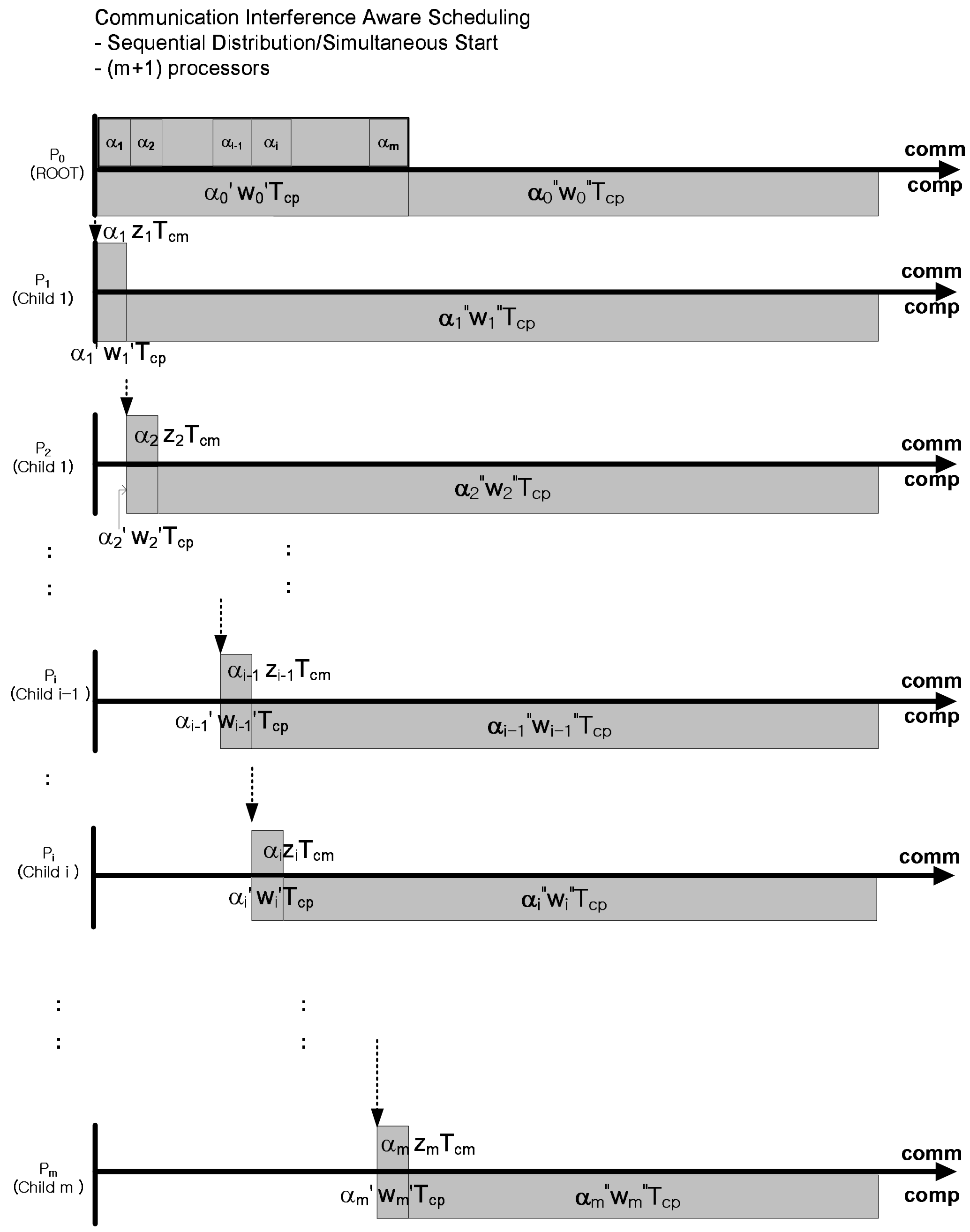


Figure 3-2: Timing diagram for IA (Interference Aware) sequential distribution and simultaneous start.

to the root processor and each child processor.

$$\sum_{i=0}^{m} \alpha_i = 1 \tag{3.1}$$

Each load fraction, α_i, for each processor is fragmented into two load fractions: one is $\alpha_i^{'}$ for the affected processor and the other $\alpha_i^{''}$ for the unaffected processor.

$$\alpha_i = \alpha_i^{'} + \alpha_i^{''} \qquad i = 0, 1, \ldots, m \tag{3.2}$$

The following equation comes from the relationship between the root processor and the first child processor. The time to compute one fragmented load $\alpha_0^{'}$ in the affected part, using $w_0^{'}$, and load fragment $\alpha_0^{''}$ in the unaffected part, using $w_0^{''}$, of the root processor should be equal to the time to distribute the load α_1 to the child processor 1 and compute one of the fragmented loads, $\alpha_1^{''}$, in the unaffected part, using $w_1^{''}$ of the child processor 1.

$$\alpha_0^{'} w_0^{'} T_{cp} + \alpha_0^{''} w_0^{''} T_{cp} = \alpha_1 z_1 T_{cm} + \alpha_1^{''} w_1^{''} T_{cp} \tag{3.3}$$

The time to compute one of the fragmented loads, $\alpha_0^{''}$ in the unaffected part, using $w_0^{''}$ of the root processor should be equal to the time to compute one of fragmented loads, $\alpha_m^{''}$ in the unaffected part, using $w_m^{''}$ of the last child processor.

$$\alpha_0^{''} w_0^{''} T_{cp} = \alpha_m^{''} w_m^{''} T_{cp} \tag{3.4}$$

The time to compute one of the fragmented loads, $\alpha_0^{'}$ in the affected part, using $w_0^{'}$ of the root processor should be equal to the time to distribute the whole load to all children processors except the load, α_0, for the root processor.

$$\alpha_0^{'} w_0^{'} T_{cp} = \left(\sum_{i=1}^{m} \alpha_i z_i \right) T_{cm} \tag{3.5}$$

The time to distribute the i-*th* load, α_i over the i-*th* link to the i-*th* processor is equal to the time to compute one of the fragmented loads, $\alpha_i^{'}$ in the affected part of the i-th processor.

$$\alpha_i z_i T_{cm} = \alpha_i^{'} w_i^{'} T_{cp} \qquad i = 1, 2, \ldots, m \tag{3.6}$$

Equation (3.7) says that the time to compute one of the fragmented loads, $\alpha_{i-1}^{''}$ in the unaffected part of the *i*−*1-th* processor is equal to the time to distribute the *i-th* load, α_i, to the *i-th* processor and compute one of the fragmented loads, $\alpha_i^{''}$ in the unaffected part of the *i-th* processor.

$$\alpha_{i-1}^{''} w_{i-1}^{''} T_{cp} = \alpha_i z_i T_{cm} + \alpha_i^{''} w_i^{''} T_{cp} \qquad i = 2, 3, \ldots, m \tag{3.7}$$

3.2.2 Analytical solution

From equations (3.2) and (3.6), one obtains

$$\alpha_i^{''} = \alpha_i - \alpha_i^{'} \qquad i = 0, 1, \ldots, m \tag{3.8}$$

$$\begin{aligned}\alpha_i^{'} &= \left(\frac{z_i T_{cm}}{w_i^{'} T_{cp}}\right)\alpha_i \\ &= k_i\alpha_i, \qquad k_i = \frac{z_i T_{cm}}{w_i^{'} T_{cp}} \qquad i = 1, 2, \ldots, m \end{aligned} \tag{3.9}$$

Substitute equation (3.9) into (3.8)

$$\alpha_i^{''} = \alpha_i - k_i\alpha_i = (1 - k_i)\alpha_i, \qquad i = 1, 2, \ldots, m \tag{3.10}$$

From (3.4) and (3.10)

$$\alpha_0^{''} = \frac{w_m^{''}}{w_0^{''}}\alpha_m^{''} = \frac{w_m^{''}}{w_0^{''}}(1 - k_m)\alpha_m, \qquad k_m = \frac{z_m T_{cm}}{w_m^{'} T_{cp}} \tag{3.11}$$

From equation (3.5)

$$\alpha_0^{'} = \frac{T_{cm}}{w_0^{'} T_{cp}}\left(\sum_{i=1}^{m}\alpha_i z_i\right) \tag{3.12}$$

Substitute equations (3.11) and (3.12) into equation (3.2)

$$\begin{aligned}\alpha_0 &= \frac{T_{cm}}{w_0^{'} T_{cp}}\left(\sum_{i=1}^{m}\alpha_i z_i\right) + \frac{w_m^{''}}{w_0^{''}}(1 - k_m)\alpha_m \\ &= c_0\left(\sum_{i=1}^{m}\alpha_i z_i\right) + r_0(1 - k_m)\alpha_m, \quad c_0 = \frac{T_{cm}}{w_0^{'} T_{cp}}, \quad r_0 = \frac{w_m^{''}}{w_0^{''}}\end{aligned} \tag{3.13}$$

In equation (3.7) we replace $\alpha_i^{''}$ with α_i by using equations (3.10) and (3.11)

$$(1-k_{i-1})\alpha_{i-1}w_{i-1}^{''}T_{cp} = \alpha_i z_i T_{cm} + (1-k_i)\alpha_i w_i^{''}T_{cp}$$

$$= [z_i T_{cm} + (1-k_i)w_i^{''}T_{cp}]\alpha_i \qquad i = 2,3,\ldots,m \tag{3.14}$$

We can simplify equation (3.14) in terms of α_i

$$\begin{aligned}\alpha_i &= \frac{(1-k_{i-1})w_{i-1}^{''}T_{cp}}{[z_i T_{cm} + (1-k_i)w_i^{''}T_{cp}]}\alpha_{i-1} \\ &= q_i\alpha_{i-1}, \qquad i = 2,3,\ldots,m \\ &= \left(\prod_{l=2}^{i} q_l\right)\alpha_1 \\ q_i &= \frac{(1-k_{i-1})w_{i-1}^{''}T_{cp}}{z_i T_{cm} + (1-k_i)w_i^{''}T_{cp}}\end{aligned} \tag{3.15}$$

From equations (3.13)and (3.15) we can obtain α_0 with respect to α_1

$$\begin{aligned}\alpha_0 &= c_0\left(\sum_{i=1}^{m}\alpha_i z_i\right) + r_0(1-k_m)\alpha_m \\ &= c_0\left(\alpha_1 z_1 + \alpha_1 q_2 z_2 + +\alpha_1 q_3 q_2 z_3 \ldots + \alpha_1\prod_{l=2}^{m} q_l z_m\right) + r_0(1-k_m)\alpha_1\prod_{l=2}^{m} q_l \\ &= \left[c_0\left(z_1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)z_i\right) + r_0(1-k_m)\prod_{l=2}^{m} q_l\right]\alpha_1 \\ &= p_0\alpha_1 \\ p_0 &= c_0\left(z_1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)z_i\right) + r_0(1-k_m)\prod_{l=2}^{m} q_l\end{aligned} \tag{3.16}$$

Above in equation (3.16), we assume all of links speed are identical each other,

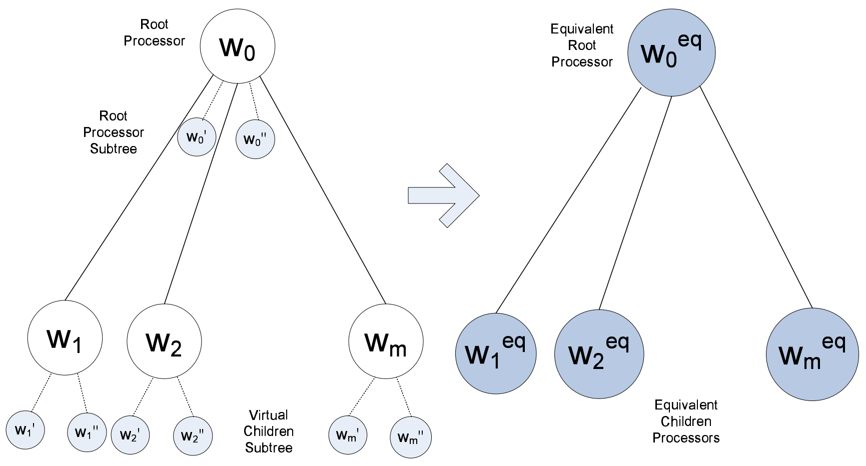


Figure 3-3: The equivalent processor concept for IA (interference aware) scheduling.

which means $z_i = z$, $(i = 1, 2, \ldots, m)$. Then p_0 becomes $p_0 = c_0 \left(1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)\right) z + r_0(1 - k_m) \prod_{l=2}^{m} q_l$. The normalization equation also leads to

$$\begin{aligned} p_0\alpha_1 + \alpha_1 + \left(\sum_{i=2}^{m}\prod_{l=2}^{i} q_l\right)\alpha_1 &= 1 \\ \alpha_1 &= \frac{1}{p_0 + 1 + \left(\sum_{i=2}^{m}\prod_{l=2}^{i} q_l\right)} \end{aligned} \tag{3.17}$$

- ***Equivalent processor in a heterogenous tree topology***

The virtual nodes of each processor are merged into one node which is identical to the combined value of two virtual nodes of each processor with respect to processing capability, as illustrated in Fig. 3-3. From the perspective of processing time, the sum of the affected and unaffected processing capability of each processor is equal to that of each equivalent processor, as the timing diagram in Fig. 3-4 indicates. This means that each virtual processor tree is replaced with the single equivalent processor, w_i^{eq} so that we can refine the closed-form solution and calculate the speedup for the whole complex network. Here $w_i^{'}$, inverse computing speed, is affected by the interference

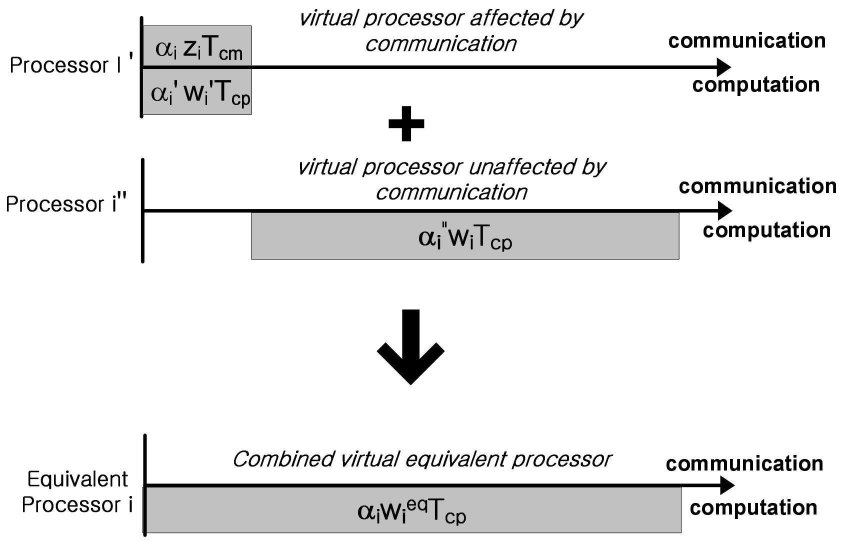


Figure 3-4: Timing diagram of the equivalent processor for Interference Aware sequential distribution and simultaneous start scheduling.

of communication, $w_i^{''}$ is not affected for the ith processor.

From the timing diagram we can obtain

$$\alpha_i w_i^{eq} T_{cp} = \alpha_i^{'} w_i^{'} T_{cp} + \alpha_i^{''} w_i^{''} T_{cp} \tag{3.18}$$

Substitute equation (3.9) and (3.10) into the above equation (3.18), so that

$$\alpha_i w_i^{eq} = k_i \alpha_i w_i^{'} + (1 - k_i) \alpha_i w_i^{''},$$

$$w_i^{eq} = k_i w_i^{'} + (1 - k_i) w_i^{''}, \qquad i = 1, 2, \ldots, m \tag{3.19}$$

Finally, we can obtain the closed-form solution for the finish time from the perspective of the equivalent processor by substituting α_1.

Finish time is

$$\begin{aligned} T_{f,m} &= \alpha_1 w_1^{eq} T_{cp} \\ &= \alpha_1 [k_1 w_1^{'} + (1-k_1) w_1^{''}] T_{cp} \\ &= \frac{[k_1 w_1^{'} + (1-k_1) w_1^{''}]}{\left[p_0 + 1 + \left(\sum_{i=2}^{m} \prod_{l=2}^{i} q_l\right)\right]} T_{cp} \end{aligned} \tag{3.20}$$

The speed-up is

$$\begin{aligned} Speedup = \frac{T_{f,0}}{T_{f,m}} &= \frac{w_0 T_{cp}}{\alpha_1 w_1^{eq} T_{cp}} = \frac{w_0}{w_1^{eq}} \cdot \frac{1}{\alpha_1} \\ &= \frac{w_0}{w_1^{eq}} \cdot \left[p_0 + 1 + \left(\sum_{i=2}^{m} \prod_{l=2}^{i} q_l\right)\right] \end{aligned} \tag{3.21}$$

Equation (3.21) can be represented in the homogeneous network by the ratio of the communication delay to the computation time of a unit load, that is the parameter $\sigma = \frac{zT_{cm}}{wT_{cp}}$.

$$\sigma^{'} = \frac{zT_{cm}}{w^{'}T_{cp}}, \qquad \sigma^{''} = \frac{zT_{cm}}{w^{''}T_{cp}} \tag{3.22}$$

Then $k_i = \frac{z_i T_{cm}}{w_i^{'} T_{cp}} = \frac{zT_{cm}}{w^{'}T_{cp}}$ of equation (3.9) becomes $\sigma^{'}$ and $r_0 = \frac{w_m^{''}}{w_0^{''}} = \frac{w^{''}}{w^{''}}$ of equation (3.13) is one, because of the homogeneous network.

So an equation for q from q_i in equation (3.15) is obtained

$$\begin{aligned}
\frac{1}{q_i} &= \frac{z_i T_{cm} + (1-k_i) w_i^{''} T_{cp}}{(1-k_{i-1}) w_{i-1}^{''} T_{cp}} = \frac{z T_{cm} + (1-k) w^{''} T_{cp}}{(1-k) w^{''} T_{cp}} \\
&= 1 + \frac{1}{(1-k)} \frac{z T_{cm}}{w^{''} T_{cp}} = 1 + \frac{\sigma^{''}}{1-\sigma^{'}} = \frac{1-\sigma^{'}+\sigma^{''}}{1-\sigma^{'}} = \frac{1}{q} \\
q &= \frac{1-\sigma^{'}}{1-\sigma^{'}+\sigma^{''}} \qquad (3.23)
\end{aligned}$$

Equation p_0 of equation (3.16) is

$$\begin{aligned}
p_0 &= \frac{z T_{cm}}{w^{'} T_{cp}} \left(1 + \sum_{i=2}^{m} (\prod_{l=2}^{i} q_l) \right) + 1 \cdot (1-k_m) \prod_{l=2}^{m} q_l \\
&= \sigma^{'} \left(\frac{1-q^m}{1-q} \right) + (1-\sigma^{'}) q^{m-1} \qquad (3.24)
\end{aligned}$$

The speedup in terms of $\sigma^{'}$ and $\sigma^{''}$ is

$$\begin{aligned}
Speedup &= \frac{w_0}{w_1^{eq}} \cdot \left[\sigma^{'} \left(\frac{1-q^m}{1-q} \right) + (1-\sigma^{'}) q^{m-1} + \frac{1-q^m}{1-q} \right] \\
&= \frac{w_0}{w_1^{eq}} \cdot \left[(\sigma^{'}+1) \left(\frac{1-q^m}{1-q} \right) + (1-\sigma^{'}) q^{m-1} \right] \qquad (3.25)
\end{aligned}$$

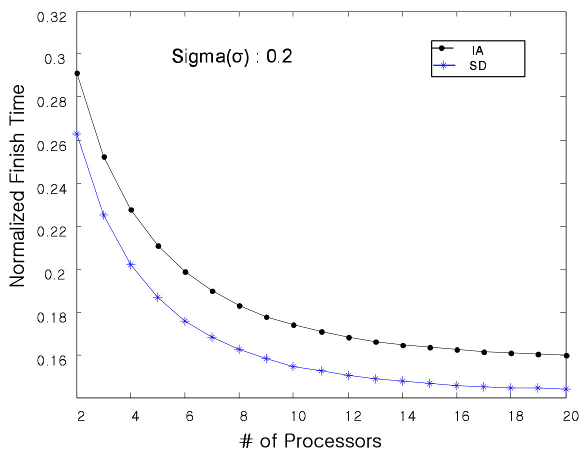
3.3 Numerical Solution and comparison

3.3.1 Numerical Solution Parameters

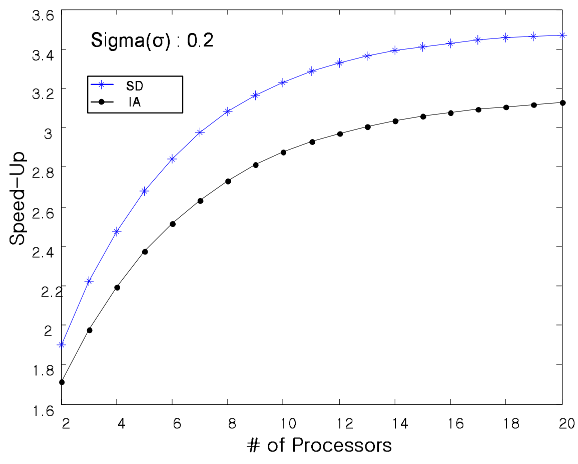
In this section, we assume a homogenous network in which each child processor has the same computing and link speed, and the root processor can have a different speed in order to focus on modeling the interference of communication. The parameters for this experiment, chosen for illustrative purposes, are as follows. In all experiments, the constants for the computation intensity, T_{cp} and communication intensity, T_{cm} is set to one. The inverse value of the root processing speed, w_0 is set to one, but the inverse values of the children processors' speeds, w_i are set to two, which means each child processor has half as much computing speed as the root processor. The degradations of computing speed for sending and receiving load are modeled as two third and half as much as the initial value respectively. The inverse communication speed, z in the network is set to 0.5 in measuring the finish time and speed-up (Fig 3-5) and is varied from 0.1 to 1.0 in investigating the effect of communication speed on computation (Fig 3-6).

3.3.2 Numerical Solution Results

Finish Time : The plot for finish time (makespan) in Fig. 3-5 (a) shows that both IA (Interference Aware) and SD (Sequential Distribution) scheduling have a similar pattern from the point of view of the finish time as the number of processors increases from 2 to 20 in Fig. 3-5 (a). As the number of processors increases 2 to 20, the finish time of the IA method is around 90% of the SD method.



(a) Finish time

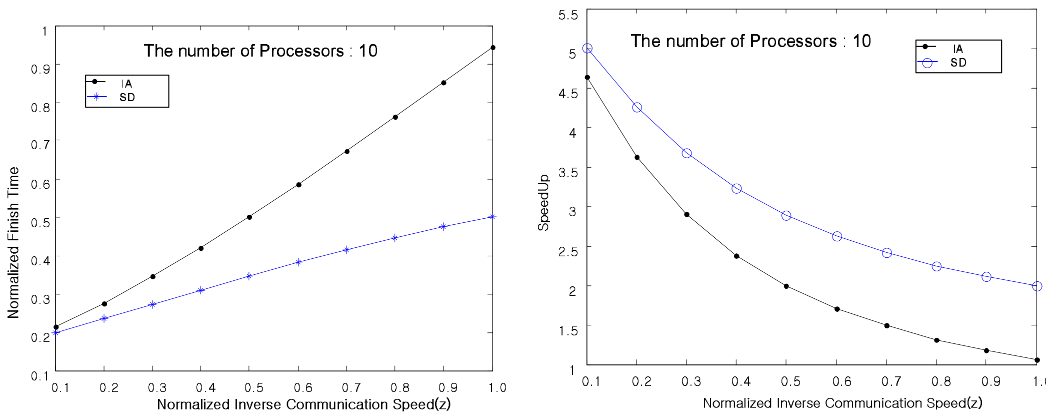


(b) Speed up

Figure 3-5: The IA (Interference Aware) scheduling versus the SD (Sequential Distribution) Scheduling.

Speedup : When we plot the speedup of both IA and SD scheduling in Fig. 3-5 (b), as the number of children processors increases, both of the speedups are saturated to some level. In general, the plot of SD is more steep than that of IA scheduling as the number of processors is varied from 2 to 10. Specifically, the number of processors that saturation occurs at is 10 in IA scheduling, while it is 15 in SD scheduling. Also IA scheduling reaches saturation earlier than SD scheduling.

Communication Speed : The plots in Fig. 3-6 demonstrate what happens to the finish time when the number of processors is fixed to 10 and the inverse value of communication speed increases from 0.1 to 1.0 gradually. We can see that the IA method is more sensitive to the effect of communication speed than the SD method. Intuitively this is apparent because the IA method considers the interference of communication on computation, one of important factors in the finish time. Meanwhile, we can observe that as the inverse constant value of communication speed, z increases, speed-ups of both the IA and SD methods drop down drastically. For the IA method, when the inverse value of communication speed is large, that is, communication speed is slow, the speed-up is very low.



(a) Finish time (makespan) vs. communication speed

(b) Speed up vs. communication speed

Figure 3-6: The IA (Interference aware) scheduling method versus than the SD (Sequential Distribution) method in terms of communication speed.

Interference aware scheduling produces a schedule with a longer finish time and less speed-up than traditional sequential distribution mechanism, because of the degradation of computing power due to communication interference. However, this result is more realistic, since it reflects the existing effect of communication on computing power in practical computing world.

3.4 Interference Aware Scheduling In Parallel Video Processing

Here we propose the IA-COMP (Interference Aware COMP) modeling method considering the interference between computation and communication in parallel video processing. In the IA-COMP method when computation and communication occur in the root processor at the same time the computing speed degrades [17]. In reflecting this assumption, we separate the computing speed into three categories: two are for sending / receiving load, and the third is for computing only. When we compare the IA-COMP method with the SD-COMP method, we see that the throughput of the IA-COMP method is somewhat less than that of the SD-COMP. Intuitively this result is apparent because the IA-COMP method has degradation on the computing speed due to the interference of communication. However it is meaningful that the IA-COMP method is closer to accurately modeling the real world environment.

3.4.1 Notation for the IA method

The variables used for modeling the impact of communication are added in this chapter.

α_i : The load fraction assigned to the ith link-processor pair (where $i = 0, 1, 2, \ldots, m$).

α_i' : The load fraction assigned to the ith link-processor pair when sending load (where $i = 0, 1, 2, \ldots, m$).

α_i'' : The load fraction assigned to the ith link-processor pair when neither sending nor receiving load (where $i = 0, 1, 2, \ldots, m$).

$\alpha_i^{'''}$: The load fraction assigned to the ith link-processor pair when receiving load (where $i = 0, 1, 2, \ldots, m$).

w_i : The inverse computing speed at the ith processor (where $i = 0, 1, 2, \ldots, m$).

$w_i^{'}$: The inverse computing speed at the ith processor when the load is sent (where $i = 0, 1, 2, \ldots, m$).

$w_i^{''}$: The inverse computing speed at the ith processor which is not affected by the communication (where $i = 0, 1, 2, \ldots, m$).

$w_i^{'''}$: The inverse computing speed at the ith processor when the load is received (where $i = 0, 1, 2, \ldots, m$).

w_i^{eq} : The equivalent inverse computing speed at the equivalent node, $node_{<i>}^{eq}$, collapsed from a single level tree rooted at $node_{<i>}$ (where $i = 0, 1, 2, \ldots, m$).

z_i : The inverse communication speed on the ith link (where $i = 0, 1, 2, \ldots, m$).

T_{cp} : Computing intensity constant.

T_{cm} : Communication intensity constant.

$T_{f,m}$: The finish time. Time at which each processor completes computation.

3.4.2 Interference aware scheduling (IA-COMP)

In this scenario, IA-COMP(Interference aware with computation on the root processor), we consider the interference to the computing speed of the root processor during communication with its children processors [18]. When the root processor computes the load assigned to itself, it experiences degradation of the computing speed due to the internal interference of distribution and collection to/from the children processors. The computing speed with communication is different from that without it. The inverse computing speed of the root processor is broken into three parts, $w_0^{'}$ for sending load, $w_0^{''}$ for no communication, and $w_0^{'''}$ for receiving a result. In addition, the load, α_0 for the root processor should also be broken into $\alpha_0^{'}$, $\alpha_0^{''}$, and $\alpha_0^{'''}$ in a similar manner (see below).

In general, we assume that the computing speed of the root processor can be different from that of each child processor, while all of the children are identical in terms of both computing and link speed, in order to compare its performance with other methods under the same parameters. From the timing diagram in Fig. 3-7, the equations for the IA-COMP scheduling method are obtained as follows.

The load fraction assigned to the root node, α_0 is broken into three fractions: the first is $\alpha_0^{'}$ for the affected inverse computing speed in sending load, $w_0^{'}$ and the second is $\alpha_0^{''}$ for the unaffected inverse computing speed, $w_0^{''}$, and the third is $\alpha_0^{'''}$ for the affected computing speed when receiving results, $w_0^{'''}$. One has :

$$\alpha_0 = \alpha_0^{'} + \alpha_0^{''} + \alpha_0^{'''} \tag{3.26}$$

$$\alpha_0^{'} w_0^{'} T_{cp} = \alpha_1 z_1 T_{cm} = \ldots = \alpha_m z_m T_{cm} \tag{3.27}$$

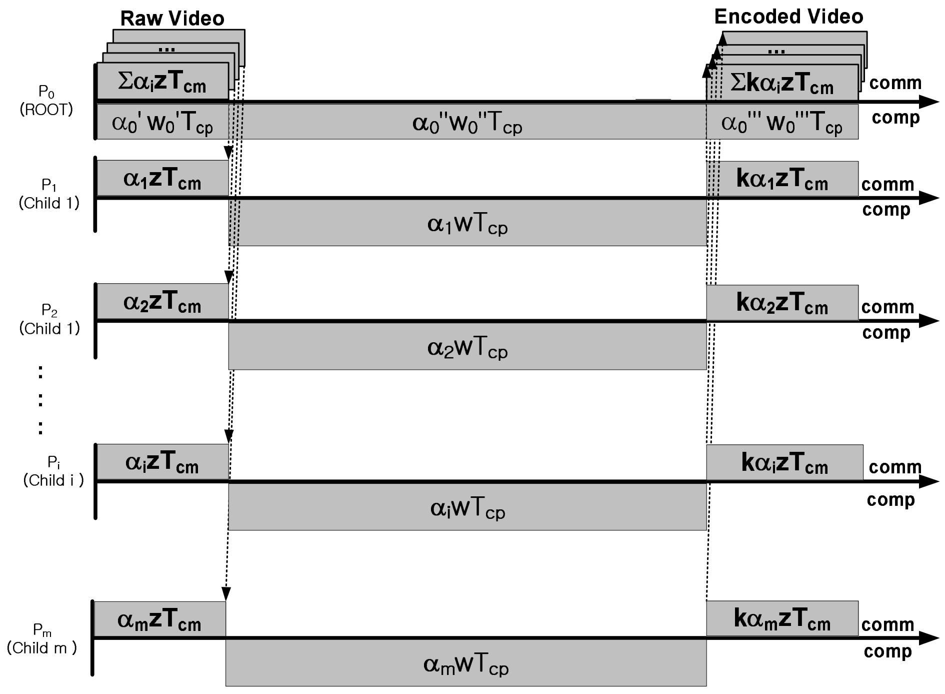


Figure 3-7: The timing diagram for the IA-COMP(Interference Aware scheduling with computation the root processor) method.

$$\alpha_0^{'} = \frac{z_i T_{cm}}{w_0^{'} T_{cp}} \alpha_i, \qquad i = 1, 2, \ldots m \tag{3.28}$$

$$\alpha_0^{''} w_0^{''} T_{cp} = \alpha_1 w_1 T_{cp} = \ldots = \alpha_m w_m T_{cp} \tag{3.29}$$

$$\alpha_0^{''} = \frac{w_i}{w_0^{''}} \alpha_i, \qquad i = 1, 2, \ldots m \tag{3.30}$$

$$\alpha_0^{'''} w_0^{'''} T_{cp} = \alpha_1 k z_1 T_{cm} = \ldots = \alpha_m k z_m T_{cm} \tag{3.31}$$

$$\alpha_0^{'''} = \frac{kz_iT_{cm}}{w_0^{'''}T_{cp}}\alpha_i, \qquad i = 1, 2, \ldots m \tag{3.32}$$

As for the root processor, the computing speed can be greater or smaller than that of the children processors, which means that the inverse value of the root processor remains the original value, w_0. The inverse computing speed of children processors is represented as $w_1 = w_2 = \ldots = w_m = w$, and the inverse link speed of children processors as $z_1 = z_2 = \ldots = z_m = z$, because of homogeneous computing and link speed. We can simplify the equations as follows:
From equation (3.27)

$$\alpha_1 = \alpha_2 = \ldots = \alpha_m \tag{3.33}$$

From equation (3.28), (3.30), and (3.32)

$$\alpha_0^{'} = \sigma^{'}\alpha_i, \qquad \sigma^{'} = \frac{z_iT_{cm}}{w_0^{'}T_{cp}}, \qquad i = 1, 2, \ldots m \tag{3.34}$$

$$\alpha_0^{''} = \rho\alpha_i, \qquad \rho = \frac{w_i}{w_0^{''}}, \qquad i = 1, 2, \ldots m \tag{3.35}$$

$$\alpha_0^{'''} = \sigma^{'''}\alpha_i, \qquad \sigma^{'''} = \frac{kz_iT_{cm}}{w_0^{'''}T_{cp}}, \qquad i = 1, 2, \ldots m \tag{3.36}$$

From equation (3.26) we can obtain for i =1

$$\alpha_0 = \sigma^{'}\alpha_1 + \rho\alpha_1 + \sigma^{'''}\alpha_1 = (\sigma^{'} + \rho + \sigma^{'''})\alpha_1 = \gamma\alpha_1 \tag{3.37}$$

Substituting equation (3.37) into the normalization equation (2.12),

$$\gamma\alpha_1 + m\alpha_1 = 1$$

$$\alpha_1 = \frac{1}{m+\gamma} \tag{3.38}$$

From the timing diagram Fig. 3-7, the finish time with $m+1$ processors, $T_{f,m}$ is

$$T_{f,m} = \alpha_0 w_0 T_{cp} = \gamma\alpha_1 w_0 T_{cp} = \frac{\gamma}{m+\gamma} w_0 T_{cp} \tag{3.39}$$

The finish time on a single root processor, $T_{f,0}$ is

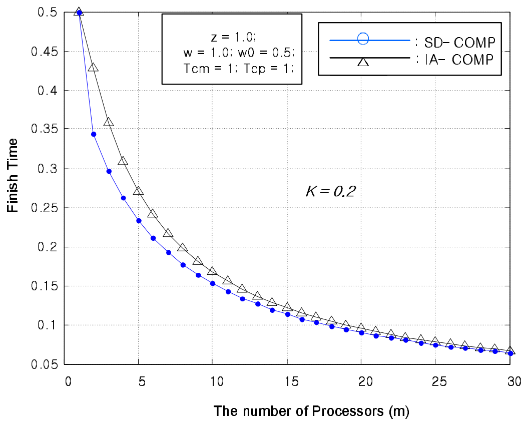
$$T_{f,0} = \alpha_0 w_0 T_{cp} = w_0 T_{cp} \qquad \alpha_0 = 1 \tag{3.40}$$

The speedup is

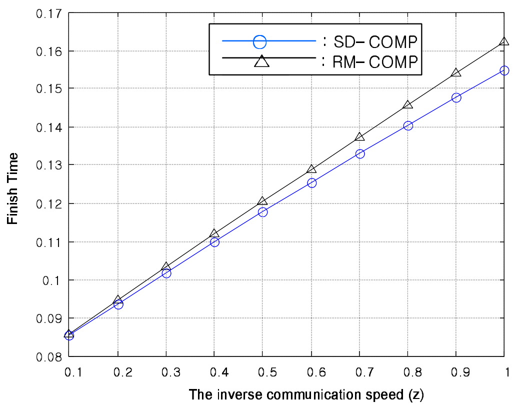
$$Speedup = \frac{T_{f,0}}{T_{f,m}} = \frac{w_0 T_{cp}}{\left(\frac{\gamma}{m+\gamma}\right) w_0 T_{cp}} = \frac{m+\gamma}{\gamma} \tag{3.41}$$

3.4.3 Comparison of the SD-COMP and IA-COMP method

In this section, we compare only two scheduling methods, SD-COMP and IA-COMP (Interference Aware scheduling with Computation), because for the SD-NO method



(a) The finish time



(b) The ratio of SD-COMP to IA-COMP method

Figure 3-8: The SD-COMP scheduling method versus IA-COMP method.

computation does not overlap with communication. We assume that the computing speed of the root processor can be different from that of each child processor and the root processor does computation and communication (distribution and collection) at the same time.

In Fig. 3-8 (a), the parameters are set up as follows: the number of processors increases 1 to 30 and the computing speed of the root processor is twice as fast as that of each child processor. When computation and communication overlap each other in the root processor, we assume that the computing speed is reduced to half of the original value in sending, and one third of the original value in receiving from all children processors. This means that the inverse computing speed, w_0', becomes twice as large when sending, and w_0''', three times as large when receiving as the original inverse computing speed of the root processor.

As we expect, the result we obtain is that the finish time of the IA-COMP scheduling method is a bit longer than that of the SD-COMP method. This is because of the interference of distribution and collection on computing speed of the root processor. As the number of processors increase linearly, the difference between them becomes small, because only the root processor suffers from the interference of communication on computation. From another perspective, the ratio of SD-COMP to IA-COMP is 92% for two processors as well as 96% for 30 processors in Table 3.1.

In Fig. 3-8 (b), we set the number of processors as 12 and make the communication speed decrease, which means that the inverse of the communication speed, z increases from 0.1 up to 1.0. As the communication speed decreases 10 times, the processing time of both the SD-COMP and the IA-COMP methods is twice as large as the starting value. Table 3.2 shows that the ratio of SD-COMP to IA-COMP is also decreased as the link speed is decreased proportionally, because the IA-COMP is more affected than SD-COMP by the communication speed.

# of Processors	2	5	10	12	20	30
IA-COMP	0.3736	0.2585	0.1675	0.1454	0.0951	0.0664
SD-COMP	0.3438	0.2340	0.1528	0.1342	0.0902	0.0640
Ratio of SD to IA	92%	91%	91%	92%	95%	96%

Table 3.1: The ratio of SD-COMP to IA-COMP method in terms of finish time for the number of processors.

z	0.2	0.4	0.6	0.8	1.0
IA-COMP	0.09456	0.11190	0.12894	0.14569	0.16216
SD-COMP	0.09366	0.10979	0.12536	0.14040	0.15493
Ratio of SD to IA	99.04%	998.11%	97.22%	96.37%	95.54%

Table 3.2: The ratio of SD-COMP to IA-COMP method in terms of finish time with communication speed decreased.

Finally, we apply a scheduling method considering the influence of communication on processor computation to parallel video processing to obtain more realistic and accurate modeling.

3.5 Conclusions

Based on this work the following conclusions can be drawn:

• Interference aware scheduling can be analytically modeled. Optimal load allocations, speedup and makespan can be found in closed form.

• Interference aware scheduling produces schedules with a larger makespan and smaller speedup than modeling that does not take communication interference into account.

• Modeling interference is realistic but is only one aspect of computer modeling. Other factors that affect results that have been considered in the literature separately include start up costs, memory hierarchy, the inclusion of front end processors and the choice of scheduling policy.

• The results and policies here are meant to be representative. They can be extended to other scheduling policies and interconnection topologies.

All of this indicates that more realistic models of computers operating in networks and grids are obtainable with some effort.

Chapter 4

Multi-source scheduling

4.1 Motivation

In very large heterogeneous networks, it will be increasingly common that multiple source nodes create and originate large amounts of data (workload) and any sink node receives data from one of a number of source nodes. For high energy and nuclear physics experiments, large amounts of data originate from distant experiments. Such data requires a high computational power and network based computing platforms in these types of experiments. Minimizing the time to process workload originated from various sources presents a great challenge that could give rise to a range of new applications.

So far research in this area includes [23] where tasks arrive according to a basic stochastic process to multiple nodes and [24] which presents a first step technique for scheduling divisible loads from multiple sources to multiple sinks, with and without buffer capacity constraints.

In [25], optimal solutions were presented for single-round model and asymptotical optimal solutions for a multi-round model were obtained in scheduling divisible workloads in a heterogeneous network. Two cases with overlapping and no overlapping of communication and computation were considered for communication latency. However, these algorithms were targeted for scheduling workload originating from a single source node. For multiple sources more sophisticated scheduling methods are necessary to obtain optimal solutions.

In previous work [26], a method of scheduling load originating from multiple sources via concurrent communication was proposed in which all of the source nodes simultaneously distributed their load to all of the sink nodes. Even though this proposed scheduling method yielded a closed form solution, all of the source nodes are forced to have multiple ports to support concurrent communication. In terms of cost-effectiveness with respect to hardware, the scheduling method using sequential communication can be improved upon.

Heuristics to find the best processor-link pair arrangement on parallel processor were investigated in [27]. To minimize the time for all of the nodes in a network to process an entire load, a greedy method to swap an adjacent pair and another method to search neighborhoods not covered by adjacent swapping were presented. It was also shown that suboptimal solutions could be obtained for a small number of processors using monetary cost as the optimization criteria. Also in [28] other heuristics for optimal load distribution sequencing were proposed in terms of monetary cost needed to utilize computation and communication. By swapping the position of two logically adjacent processors, the optimal load distribution sequence was found to improve computation and communication cost for a single level heterogeneous tree network. Specifically, optimal load distribution for a homogeneous bus network is shown to involve sequencing which is non-decreasing in the order of the sum of the computation and communication costs.

The paper [29] proposed a method of scheduling multiple sources and multiple sinks for efficient use of distributed resources via network partition. Also, min cost and multi-commodity flow is used in formulating a scheduling method for steady state divisible load in a linear network model. However, techniques for transient state scheduling were not extensively covered where divisible load can dynamically vary with more complicated network topologies and multiple source nodes.

Our Contribution

In this paper, a method to schedule workload from multiple source nodes is proposed by a specific network partitioning technique via the use of a genetic algorithm. It is assumed that multiple source and sink nodes are randomly scattered and connected to each other over the network. The network is fragmented into several partitions matching the number of source nodes. Computation in each partition is independently performed on single source and multiple sink nodes. That is, a complicated problem for multi-source scheduling is simplified into several single source scheduling problems after network partitioning. In this approach, each partition is simultaneously evaluated by running an optimal single source load scheduling algorithm. Through this procedure we can obtain the effect of concurrent communication with simultaneous distribution scheduling. Sophisticated heuristic approaches such as genetic algorithms [30] can be considered in partitioning a network for this multi-source load scheduling optimization problem. In applying genetic algorithms, a new partition can be created by transferring a sink node from one partition to another. A new offspring partition can be obtained by changing the order to distribute load from a source node to the sink nodes in each partition. This procedures to create new partitions via genetic algorithms is repeated until solution time converges to a minimized value.

4.2 Multi-source scheduling in a large heterogeneous network

4.2.1 Problem formulation and definition

In a large heterogeneous network, it is very common that loads originating from multiple source nodes are distributed to multiple sink nodes. While all the source nodes can distribute loads to all the sink nodes, some of sink nodes can be divided to join a specific partition related to a source node. The goal is to minimize the time to finish all the jobs submitted from every source node. For a heterogenous network with multiple sources, source nodes are each assigned to one partition respectively. For a sink node which can be connected to multiple source nodes, we need make a decision on which source node is a good candidate as illustrated in Fig. 4-1. In doing so, we can consider several parameters such as the computing speed and communication speed in each partition.

$\alpha_{i,j}$: The load fraction assigned from the i^{th} source node to the j^{th} sink node.

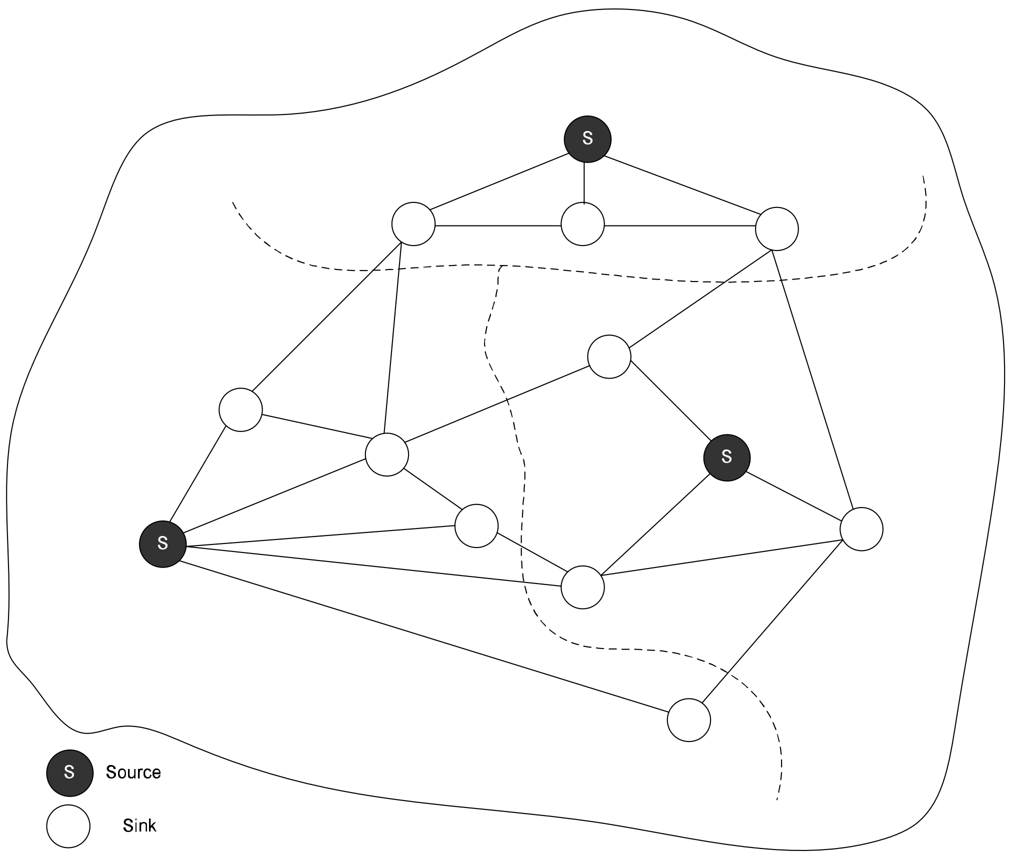


Figure 4-1: Multi-source heterogeneous network

w_i: The inverse computing speed on the i^{th} processor.

z_i: The inverse link speed of the i^{th} link.

T_{cp}: Computing intensity constant:

The entire load can be processed in $w_i T_{cp}$ seconds on the i^{th} processor.

T_{cm}: Communication intensity constant:

The entire load can be transmitted in $z_i T_{cm}$ seconds over the i^{th} link.

$T_{f,i}$: The finish time:

Time when all of the processors in the i^{th} partition complete computation.

Thus $\alpha_i z_i T_{cm}$ is the time to transmit the fraction α_i of the entire load over the ith link. Note that the units of $\alpha_i z_i T_{cm}$ are [load] $\times$ [time/load] $\times$ [dimensionless quantity] = [time]. Likewise, $\alpha_i w_i T_{cp}$ is the time to process the fraction α_i of the entire load on the ith processor. Note that the units of $\alpha_i w_i T_{cp}$ are [load] $\times$ [time/load] $\times$ [dimensionless quantity] = [time].

4.2.2 Two source scheduling

In this section, multi-source scheduling can be explained in details for the network with two source nodes and multiple sink nodes. A sink node can be assigned to one of both source nodes via network partitioning. The number of partitions via network partitioning should be the same as the number of source nodes. In Fig. 4-2, two source nodes are fully connected with seven sink nodes each other. The partition 1 is composed of one source node, three sink nodes, and the partition 2 is one source node and four sink nodes. This can easily generalized to situations where the amount of computing capability in each partition is different perhaps because of differences in the amount load generated by each source. The partition model can also be generalized to have multiple source nodes in each partition.

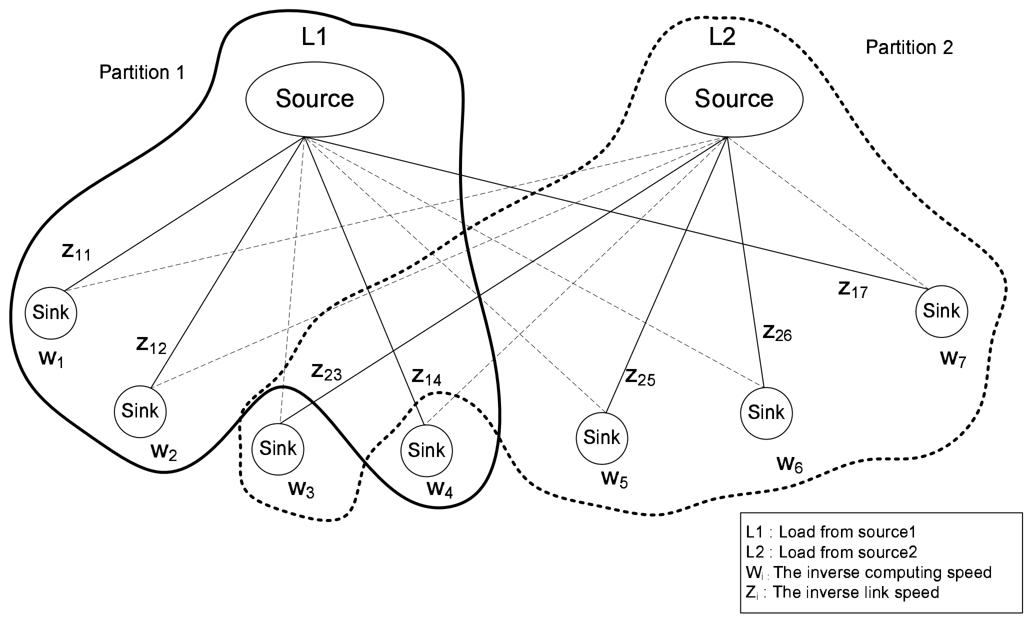


Figure 4-2: Two partitions should be created so that the sum of the computing speed $(\frac{1}{w})$ of both partitions can be similar each other if possible.

Within a partition as an example, the sequential distribution strategy is applied from the root to children nodes. The finish time of two partitions can be different, because the capability of both partitions is different in terms of the computing speed and the link speed. As illustrated in Fig. 4-3, the goal is to minimize finish time to complete sequential distribution and computation from the perspective of the entire network.

We can obtain two finish times, $T_{f,1}$, $T_{f,2}$, one for each partition respectively. Here T_w is the waiting time, the difference between finish time, $T_{f,1}$ and $T_{f,2}$ resulting from each partition. During the waiting time, T_w, one partition should wait for the other

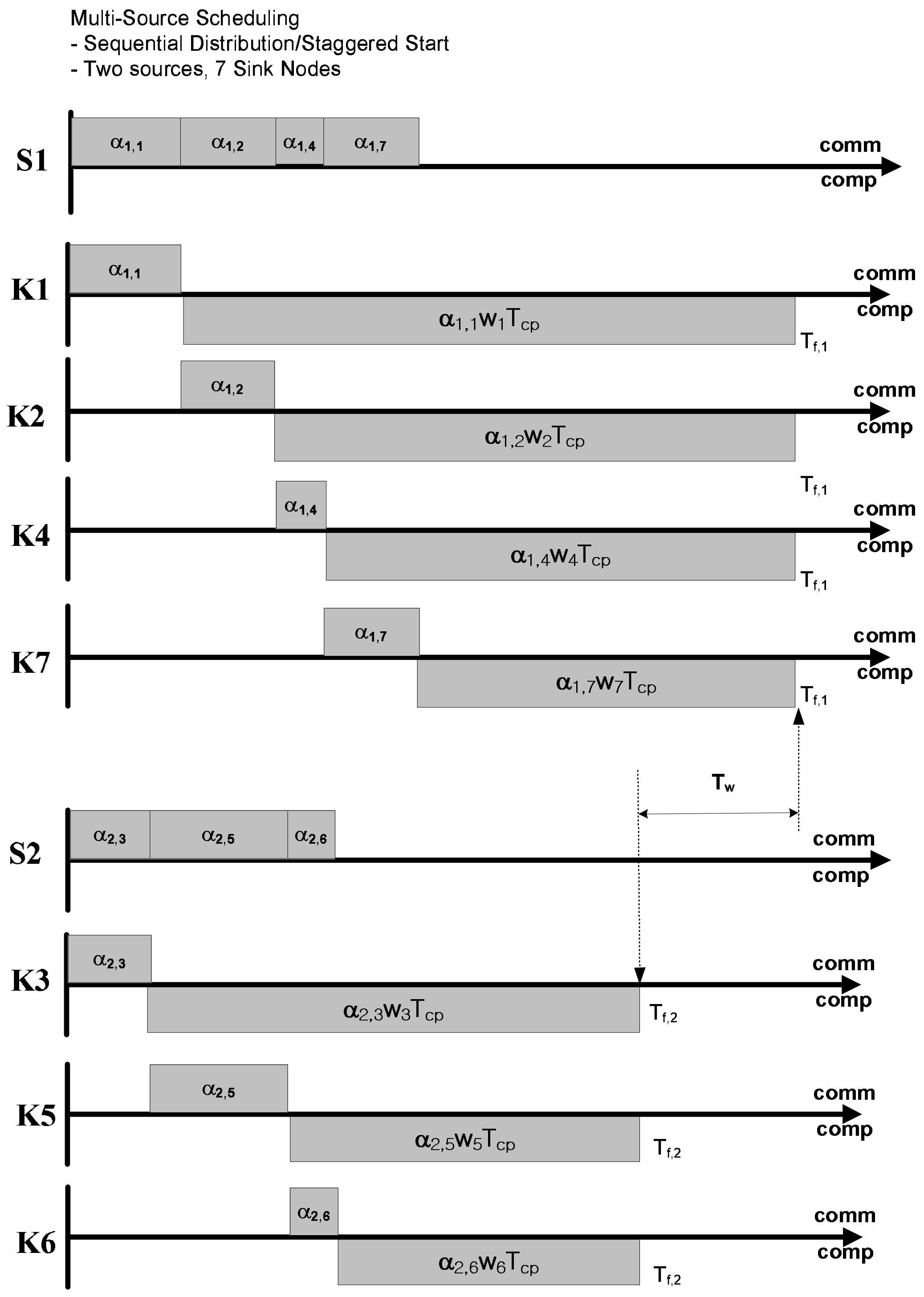


Figure 4-3: Timing diagram for multi-source scheduling with sequential distribution in a network with two source and seven sink nodes. $T_{f,1}$ and $T_{f,2}$ are the finish times and T_w is the waiting time.

partition to complete jobs. It is highly possibility that these two finish times are different, since the computing speed and the link speed of nodes are usually different in each partition. We determine the larger value among two finish times as the solution for this network configuration.

$$T_{f,1} = \alpha_{1,1} z_1 T_{cm} + \alpha_{1,2} z_2 T_{cm} + \alpha_{1,4} z_4 T_{cm} + \alpha_{1,7} z_7 T_{cm} + \alpha_{1,7} w_7 T_{cp}$$

$$T_{f,2} = \alpha_{2,3} z_3 T_{cm} + \alpha_{2,5} z_5 T_{cm} + \alpha_{2,6} z_6 T_{cm} + \alpha_{2,6} w_6 T_{cp}$$

Furthermore, we can use genetic operators such as crossover and mutation to reconstruct network partitions in order to find an optimal network partition.

4.2.3 Network partitioning

For multiple sources, we need more capable methods to schedule load distribution. There can be a method for all of the sources to distribute load to all of the sink nodes. However, we propose a network partitioning technique to schedule multi-source load distribution. This scheduling method is that the largest finish time among all the partitions is minimized from the perspective of the entire network. The entire network is fragmented into several small network partitions matching the number of source nodes based on the sum of the computing speed in a partition. We construct the sum of the computing speed in each partition to be as similar as possible. After network partitioning, all the partitions are executed independently to obtain the finish time at the same time. Within each partition, all nodes participate in computation via a traditional sequential distribution model using DLT (divisible load theory). From the results from every partitions, we can obtain the finish time. If a partition completes computation first, it waits until other partitions complete their loads. So the finish time when the last partition completes its load is selected as the final finish time for the entire network. The goal is to minimize the gap between the finish times of all the partitions, because the gap is the waiting time from the perspective of the entire network. We continue this partitioning and evaluation process until we can obtain near optimal solution time over the entire network.

- **Initial partitioning** We divide an initial randomly constructed network into smaller partitioned networks according to the computing speed of each sink node. That is, we assign a random sink node to one of the partitions, and then the others to the remaining partitions to which every source node is already assigned. When the second sink node is added into the partition which has already sink nodes, we compare the sum of the computing speed of sink nodes with those of other partitions. If the sum of computing speeds of nodes in the partition for the sink node to be added is less than those of other partitions, the sink node is added. Otherwise, the sink node should be added into the other partition which has the smallest sum of the nodal computing speeds.

- **Adaptive partitioning**

We make the total computing speed of all of the sink nodes in each partition to have a similar capability among partitions. Each partition which consists of one source and several selected sink nodes can be changed moving the member sink node between partitions.

Then the locally optimal solution time is measured for all of the partitions, and the maximum value among them is determined as a solution time for all of the partitions over the entire network. This partitioning continues to converge at a small value through this series of procedures as in Fig. 4-4.

One of the simple methods to move a sink node between partitions is to move the sink node with the smallest computing speed. It makes sense in that the sink node with the smallest computing speed has a more local impact.

4.2.4 Load distribution sequencing

In this section, the optimal sequencing to find the optimal processor-link pair is considered in a single level tree network where the single source sequentially distribute load to sink nodes on a tree network. The goal of the optimal sequencing is to find the optimal sequence to distribute load to sink nodes by swapping two processor-link pairs without any change of computing link speed and obtain the minimum finish time under the optimal sequence in a partition. Because the order of load distribution affects the finish time, sink nodes can be arranged to complete the workload, resulting in an effective increase in computing power. Heuristic methods, such as processor arrangement proposed in the previous work [28] can be used to find the optimal sequence in terms of monetary cost incurred from the use of the respective processor and link pair. That is, a current profile is kept about processor-link pair and updated every time two adjacent processor-link pairs are swapped each other, until the optimal sequence of processors is obtained. To improve the processor arrangement, several initial profiles were considered as starting points, and a neighborhood of processor is

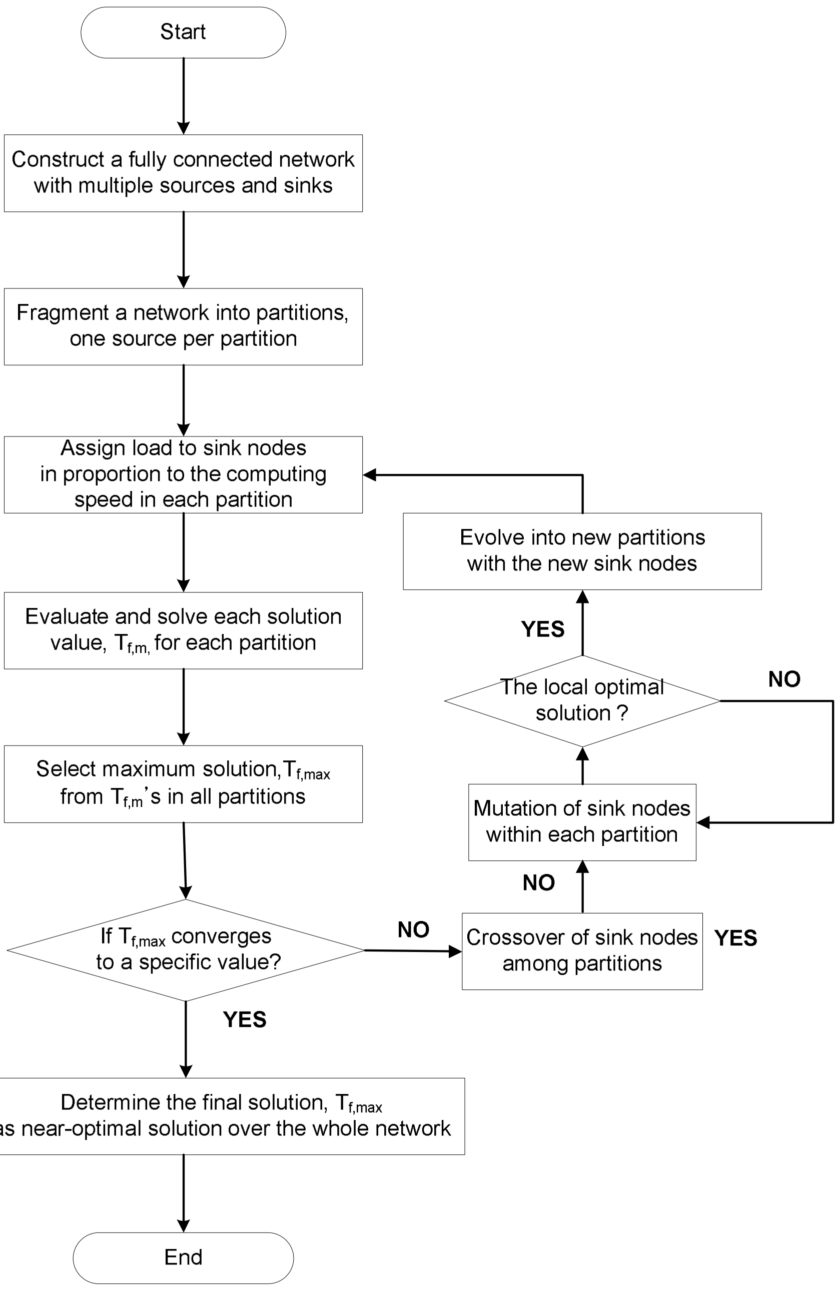

Start
Construct a fully connected network with multiple sources and sinks
Fragment a network into partitions, one source per partition
Assign load to sink nodes in proportion to the computing speed in each partition
Evaluate and solve each solution value, $T_{f,m,}$ for each partition
Select maximum solution, $T_{f,max}$ from $T_{f,m}$'s in all partitions
If $T_{f,max}$ converges to a specific value?
NO
YES
Crossover of sink nodes among partitions
YES
NO
Mutation of sink nodes within each partition
The local optimal solution ?
NO
YES
Evolve into new partitions with the new sink nodes
Determine the final solution, $T_{f,max}$ as near-optimal solution over the whole network
End

Figure 4-4: Flowchart for multi-source scheduling via genetic algorithm in grid network

extended to other pairs which are not covered by adjacent processor-link pair. In this paper, the initial profile is a network partition resulting from crossover operation. Such a network partition is used as a starting point in finding the optimal load sequencing. Also swapping algorithm of processor-link pairs is applied not only to adjacent nodes, but also to two randomly selected sink nodes.

When finding the optimal sequence in a network under the sequential load distribution scenario, the previous processor arrangement method is limited to a number of processors at 20, while our proposed optimal load sequencing goes to more than 200 processors. The reason is that the evaluating function of processor arrangement to obtain monetary cost is much more complex than ours to simply find the finish time. Another reason is computing power has advanced since the original publication [27]. So our optimal load sequencing method can be applied to a large network with the large number of processors as well as parallel systems with a small number of processors.

Algorithm 1 Multi-source scheduling algorithm

```
 1: // **Initial Population**
 2: Generate multiple sources and sink nodes, {source1,source2;sink1, sink2,...,sink_N}
 3: $src1\_comp$ , $src2\_comp \leftarrow$ rand[1,20];
 4: $src1\_comm$ , $src2\_comm \leftarrow$ rand[0.1,1.0];
 5: $sink[i]\_comp \leftarrow$ rand[1,20]; // i=1 to m
 6: $sink[i]\_comm \leftarrow$ rand[0.1,1.0]; // i=1 to m
 7: // **Initial Network Partitioning**
 8: part1.src = src1; part2.src = src2;
 9: **for** $i = 1$ to $\#Nodes$ **do**
10:     **if** $part1(\Sigma(sink[i]\_comp)) < part2(\Sigma sink[i]\_comp))$ **then**
11:         part1[n1].sink = sink[i];
12:     **else**
13:         part2[n2].sink = sink[i];
14:     **end if**
15: **end for**
16: Tfm_best = Max(part1_Tfm, part2_Tfm);
17: // **Adaptive Network Partitioning**
18: ***Crossover operator : Go to Algorithm2***
19: ***Mutation operator : Go to Algorithm2***
20: // **Selection of the fitness solution**
21: **if** Tfm_new < Tfm_best **then**
22:     Tfm_best = Tfm_new;
23: **end if**
24: //**Termination**
25: **if** (Tfm_best $\simeq \rho$) AND (T_wait_best $\ll \epsilon$) **then**
26:     terminate;
27: **end if**
```

Algorithm 2 Adaptive Network Partitioning - crossover and mutation operation

```
1: {Crossover operator}
2: for $i = 1$ to $\#Nodes$ do
3:     index1 = random[1,Nodes];
4:     index2 = random[1,Nodes];
5:     switch(part1, index1, part2, index2);
6:     Tfm_new = Max(part1_Tfm, part2_Tfm);
7:     if (Tfm_new < Tfm_best then
8:         Tfm_best= Tfm_new;
9:     end if
10: end for
11: {Mutation operator}
12: for $i = 1$ to $\#partitions$ do
13:     #PNodes = Number of sinks in a partition
14:     for $j = 1$ to $\#PNodes$ do
15:         n1 = rand[1,$\#PNodes$]
16:         n2 = rand[1,$\#PNodes$]
17:         mutate(n1, n2)
18:         Tfm_new = Finish_Time(partition$i$);
19:         if (Tfm_new < Tfm_best) then
20:             Tfm_best= Tfm_new;
21:         end if
22:     end for
23: end for
```

4.3 Multi-source scheduling via genetic algorithm

In this section, we propose a heuristic mechanism for multi-source scheduling method via genetic algorithms in which the network is fragmented into the same number of partitions as the number of source nodes.

4.3.1 Genetic algorithm heuristics

- **Representation**

The source nodes and sink nodes of each partitioned network are represented by a string of binary digits. We assume that there are the number of 'm' source nodes and the number of 'n' sink nodes in the entire network. Each partition can be shown as follows:

partition*1* = (1,0, . . . ,0:1,0,0, . . . ,1)

partition*2* = (0,1, . . . ,0:0,0,1, . . . ,0)

. . .

partition$m$ = (0,0, . . . ,1:0,1,0, . . . ,0)

The first half of the string is for the source nodes and the second half of the the string is for the sink nodes delimited by semi-colon(':'). A source node and sink nodes can participate the only one partition.

The second representation is expressed as a string sequence to distribute load from the source node to sink nodes.

Parent partition = (s1, k1, k2, k4, k7)

Offspring1 = (s1, k2, k1, k4, k7)

Offspring2 = (s1, k4, k2, k1, k7)

Offspring3 = (s1, k7, k2, k4, k1)

- **Initial population**

The initial population of M source nodes and N sink nodes are randomly generated. We can execute a genetic algorithm either with initially no partition and a fully connected network or initially with partitioned networks for the entire network.

- **Genetic operators**

Crossover operator The crossover operator is used to exchange randomly chosen sink nodes among partitions with respect to the inverse the computing speed. There exist a variety of crossover operators, which perform single crossover, or group crossover operation.

Mutation operators The mutation operator is used to change the sequence to distribute load from source node to children sink nodes within each partition. Through this mutation operator, a new partition with a different sequence of nodes can be created. The solution for this partition is also different from the previous parent partition. We can obtain the locally optimal solution for each partition as long as the size of partition is small enough to evaluate finish time for all of the combinatorial sequence of nodes.

- **Fitness function to evaluate partitions**

The following closed-form solution for finish time (makespan) of each partition with sequential distribution and simultaneous start scheduling mechanism [12, 13] is used as a fitness function in evaluate each partition

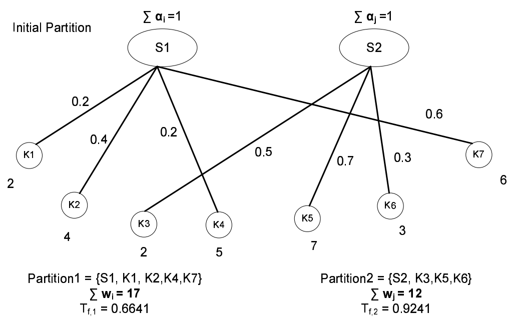
$$T_{f,m} = \frac{w_0 T_{cp}}{1 + k_1 \left[1 + \sum_{i=2}^{m} (\prod_{l=2}^{i} q_l)\right]\Big)} \tag{4.1}$$

$$k_1 = \frac{w_0}{w_1}, \qquad q_i = \frac{[w_{i-1}T_{cp} + z_{i-1}T_{cm}]}{w_i T_{cp}}, \qquad i = 2, 3, \ldots\ m$$

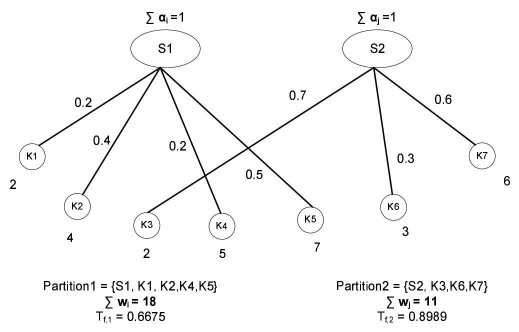
We evaluate all the partitions by this fitness function to find the maximum solution among partitions and determined the maximum solution as the value for the fittest in each round.

4.3.2 Network partitioning via crossover operator

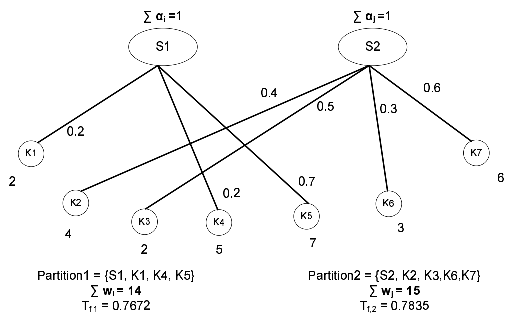
During every round of partitioning, the maximum finish time among partitions is recorded until the recorded finish time converges to a specific value. Through this heuristic approach, we can schedule load distribution originating from multiple sources to obtain a near-optimal solution. An example of scheduling procedures via genetic algorithm can be shown in Fig. 4-5 in detail. From the initial partition, we can obtain the finish time of each partition with sequential load distribution from each source node to sink nodes within a partition. The finish time from each partition is different, and so we need to schedule the members of each partition to make the sum of the total computing speed of each partition as similar to each other as possible. Otherwise load could be transferred between partitions and an improved solution would result. Using the crossover operator, the sink node K7 in partition 1 is exchanged with the sink node K5 resulting in partition 2 as Fig. 4-5 (a), (b).



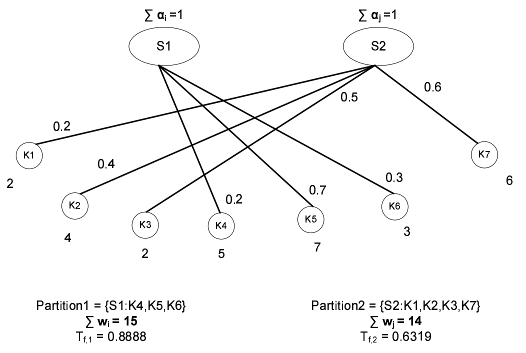
(a) Initial partition



(b) The second partition



(c) The third partition



(d) The final partition

Figure 4-5: The network partitioning via the crossover operator.

The initial Partition

partition*1* = (1,0:1,1,0,1,0,0,1)

partition*2* = (0,1:0,0,1,0,1,1,0)

The second Partition

partition*1* = (1,0:1,1,0,1,1,0,0)

partition*2* = (0,1:0,0,1,0,0,1,1)

The new partition in Fig. 4-5 (c) becomes a new offspring originating from the previous partition via the crossover operator.

The third Partition

partition*1* = (1,0:1,0,0,1,1,0,0)

partition*2* = (0,1:0,1,1,0,0,1,1)

If the crossover operation is applied enough times to converge at a specific value for each partition, a near-optimal value can be obtained. In this Fig. 4-5, the third partition results in the minimum finish time, $T_{f,m} = 0.7835$ among all the partitions.

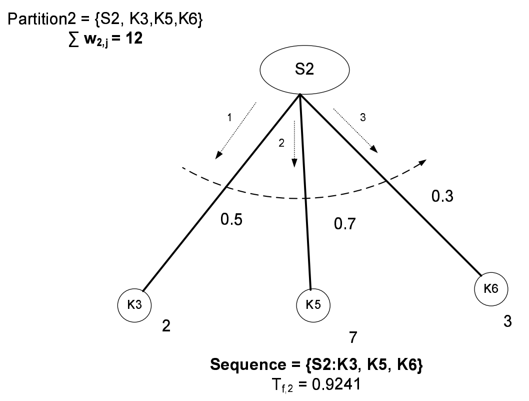
The final Partition

partition*1* = (1,0:0,0,0,1,1,1,0)

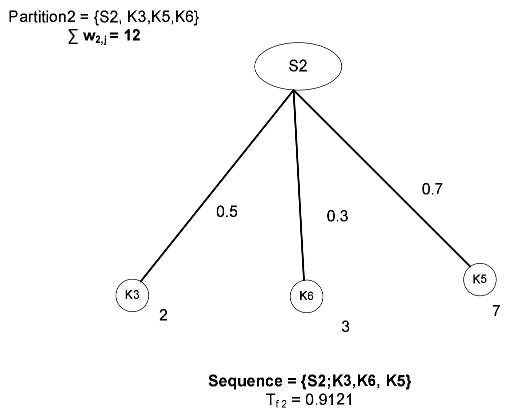
partition*2* = (0,1:1,1,1,0,0,0,1)

4.3.3 Load sequencing via mutation operator

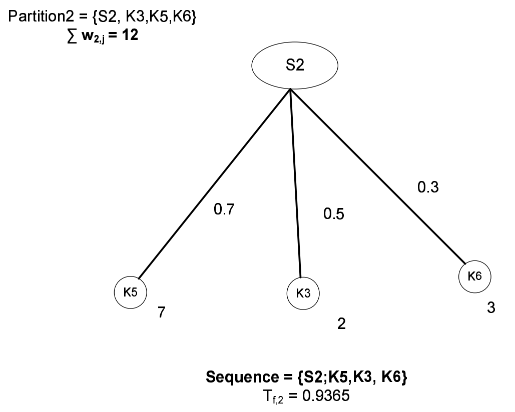
The mutation operator in this paper is used in finding the optimal sequence to distribute load from the source node to children sink nodes within each partition. If this



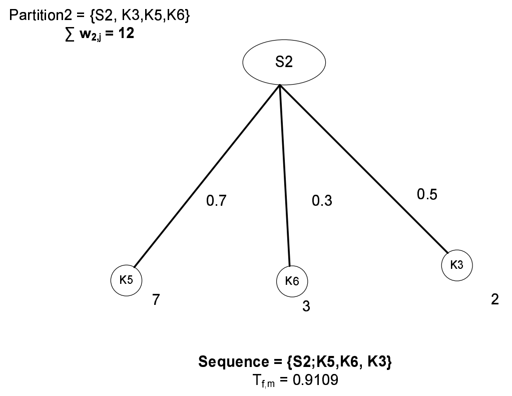
(a) Initial partition



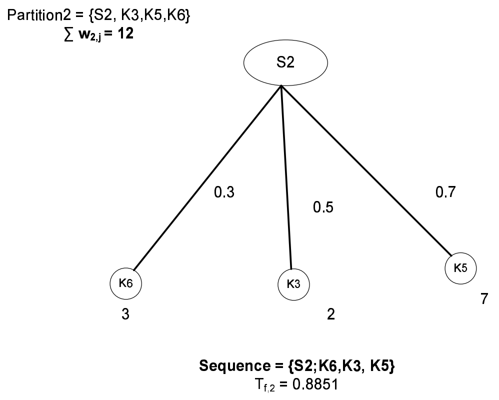
(b) The second sequence



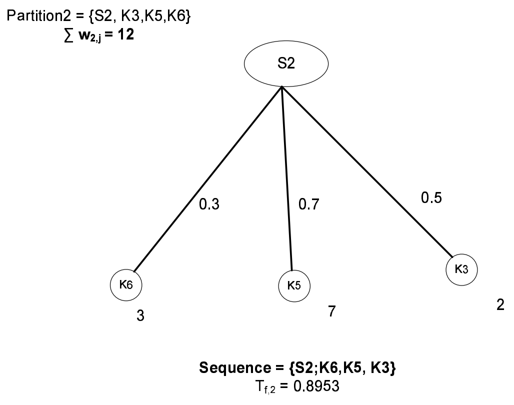
(c) The third sequence



(d) The fourth sequence



(e) The fifth sequence



(f) The sixth sequence

Figure 4-6: Node sequencing via the mutation operator.

optimal sequence is determined, we can find the local optimal solution of the finish time for each partition of the network which can be varied according to the order to distribute load to the children sink nodes. The sequence of load distribution is left to right in direction. That is, in an example, Fig. 4-6 (a), the source node, S2 distributes load first to the sink node, K3, next to K5, and last to K6. As illustrated in Fig. 4-6, when the mutation operator is applied to the initial parent partition, new offsprings with different sequences for receiving load from the source node result. Even though the sum of the computing speeds of sink nodes is kept unchanged, the finish time, $T_{f,2}$ of each offspring in the partition 2 can change according to the sequence of receiving load. The possible combinatorial sequences of sink nodes receiving load from the source node is six for this case.

Parent partition = (s2, k3, k5, k6)

Offspring1 = (s2, k3, k6, k5)

Offspring2 = (s2, k5, k3, k6)

Offspring3 = (s2, k5, k6, k3)

Offspring4 = (s2, k6, k3, k5)

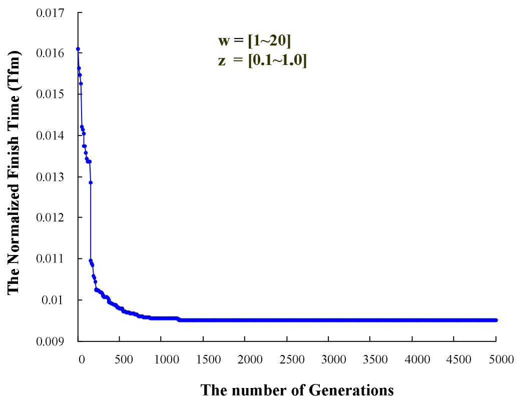
Offspring5 = (s2, k6, k5, k3)

Through the mutation operator, we can obtain the local optimal solution for sequential distribution scheduling in a single tree partitioned network. That is the minimal finish time, $T_{f,2}$ is 0.8851 for the fifth sequence in Fig. 4-6.

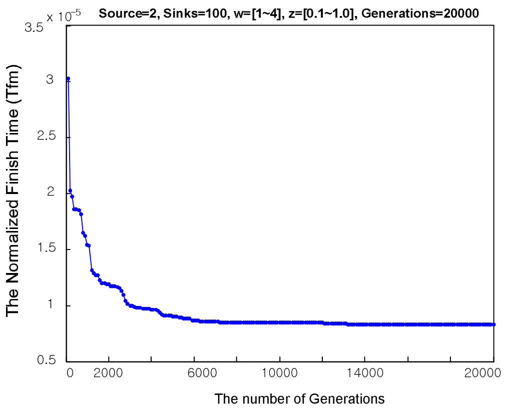
4.4 Performance analysis and comparison

In this experiment, a fully heterogenous network is considered to be applicable to more general scenarios. That is, each computing speed of source and sink nodes and each link speed among source nodes and sink nodes are different. Each inverse value of computing speed, w_i is randomly assigned within 1 to 20 and each inverse value of link speed, z_i is also randomly assigned within 0.1 to 1.0. It is assumed that source nodes are fully connected to all of the sink nodes and each source node can be connected to all sink nodes. The link speed between every source node and one sink node is also considered to be identical.

• **The finish time** In Fig. 4-7, we set the number of sink nodes to 100 and two source nodes. Each partition is evaluated by the fitness function (4.1) resulting from sequential distribution scheduling policy. After the whole network is fragmented into partitions, each sink node receives and computes load from a source node sequentially.



(a) The finish time, w=(1,20), z=(0.1,1.0)



(b) The finish time, w=(1,4), z=(0.1,1.0)

Figure 4-7: The near-optimal solutions for the finish time, $T_{f,m}$

In Fig. 4-7, the solution for the finish time and the waiting time are gradually converged to a specific value. One of the remarkable results from our proposed method is that we can obtain the near-optimal solution for the finish time sooner than expected. Every generation the finish times from each partition are calculated, and larger value is selected as solution, since a partition with small finish time should wait for the other partition to complete a job. The finish time converges to the specific small value after around 1500 generations. At the point we can also obtain the minimum waiting time between partition 1 and 2. The other result is node scalability which means that we can increase the number of sink nodes from 10 to hundreds, such as 200, with two source nodes, while our proposed network partitioning algorithms are converged to the near-optimal solution with algorithm execution time around 1 minute. Through these results, our proposed network partitioning scheduling is shown to be very scalable in terms of the number of processors as well as fast in the algorithm execution time.

In Fig. 4-7, we can see that the finish time varies every time the crossover operation is applied to the partition 1 and 2. From these values of the finish time the only better value is selected and recorded being compared to the previous best value. After a solution is obtained through this crossover operation, the mutation operator is applied to the partition. Within a partition with fixed sink nodes, a better solution is found by swapping node sequence. In reality, a better solution for the partition resulting from the crossover operation is found, since the sequence of load distribution can affect the finish time. Through these two operators, crossover and mutation, a near-optimal solution close to optimal solution is obtained.

- **The waiting time**

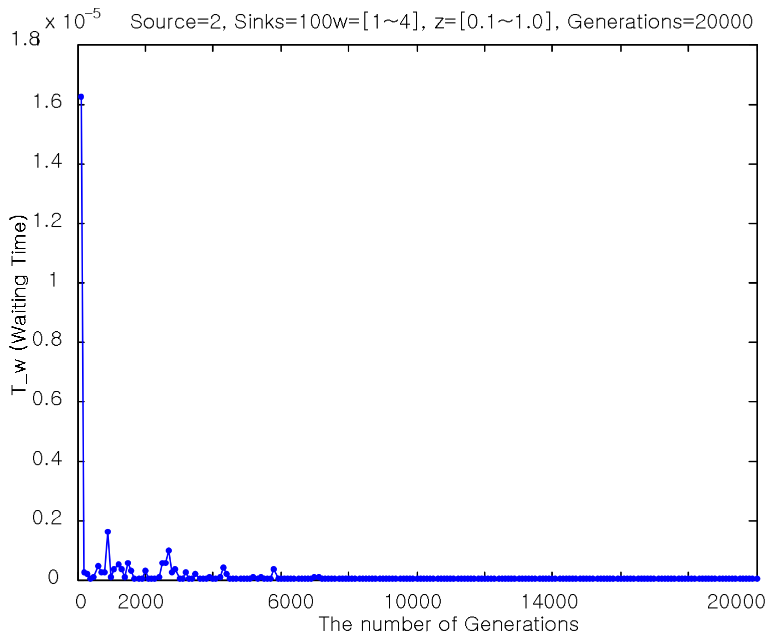


Figure 4-8: The near-optimal solutions for the waiting time, T_w

In Fig. 4-8, the waiting time is varied along with the evaluation of partitions. Even though the finish time decreases to a small value, the reason for the waiting time to go up and down is that the waiting time is the difference between two partitions. If one partition has a very small value and the other partition has a somewhat larger value, the gap can be bigger than the previous values.

4.5 Conclusion

In this paper, sophisticated heuristics for scheduling workloads originating from multiple source nodes in large heterogeneous networks are proposed. Through association of network partitioning and genetic algorithms we can obtain a feasible solution for a combinatorial problem and suggest a more cost-effective scheduling method compared to the concurrent scheduling method based on concurrent communication for multi-source scheduling.

Chapter 5

Real-time scheduling

It has been increasingly important to provide performance guarantees to deadline-constrained jobs originating from large scale experiments. To efficiently cope with these heavy workloads, divisible load theory (DLT) with real-time characteristics has been recently important in a heterogeneous Grid/cluster systems, where the job with the earliest deadline is first scheduled. It is a major trend that the cluster system is also heterogeneous in the computing capability as well as in the link speed, so-called, heterogeneous cluster systems [?]. The previous work is only limited to a homogeneous cluster environment [31] and different available processing time is considered in [32, 33], a fully heterogeneous system was not modeled in the sense of schedulability yet.

For example, the ATLAS (A Toroidal LHC ApparatuS) is one of the particle detector experiments using the Large Hadron Collider (LHC), at the European Organization for Nuclear Research (CERN) in Switzerland, where CERN, the tier 0 site generates the original experimental data and couple of tier 1 sites including BNL archive and distribute the data in each site to a couple of universities in the tier 2 site so that the universities can analyze the data [9]. From the BNL point of view, nine institutes in tier 2 of the ATLAS project, such as University of Michigan, SLAC (Stanford Linear Accelerator Center), and Boston University, should be connected through a different WAN (wide area network), such as, ESnet, Ultralight, or Internet2 to BNL (Brookhaven National Laboratory) as shown in Fig 5-1. In this configuration, each site can have heterogeneous cluster systems with different computing capability and there is a heterogeneous computing and link capability *within* each cluster system. Specifically, the hardware of the RACF (RHIC ATLAS Computing Facility) of the STAR project in BNL consists of a combination of commodity-based processing servers, enterprise-class UNIX servers and highly-specialized mass storage systems connected together by a high-speed network infrastructure. The RACF is an example of heterogeneous system with currently over 4000 processors and a distributed and centralized disk storage farm (1 PB of on-line disk storage).

After the RHIC (Relativistic Heavy-Ion Collider) at Brookhaven National Laboratory came on-line in 1999, the Solenoidal Tracker At RHIC (STAR) experiment began data taking and concurrent data analysis that will last about ten years. STAR needs to perform data acquisition and analyzes over approximately 250 tera-bytes of raw data, 1 peta-bytes of derived and reconstructed data per year. Such data in STAR experiments require bulk file transfers between heterogeneous systems in sites within a limited time over heterogeneous network. To satisfy the deadline of each job, sophisticated scheduling algorithms with respect to deadlines are needed. In [24, 34], all of these recently emerging platforms require a sophisticated scheduling

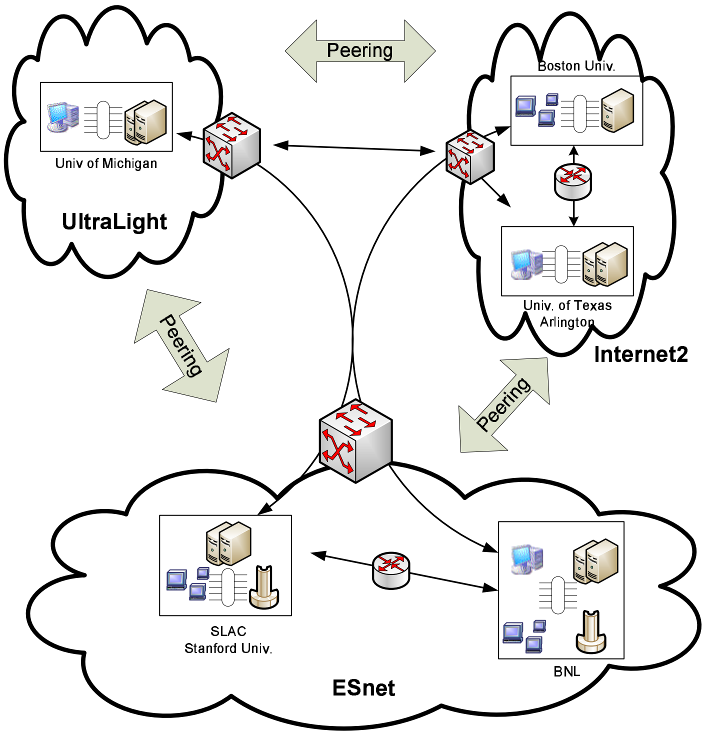


Figure 5-1: A heterogeneous Grid/cluster system example, TeraPaths network configuration of the ATLAS project.

strategy to efficiently make use of distributed computers, high-speed networks and storage resources in terms of deadlines and scheduling methods for a data intensive Grid operating under capacity constraints.

In [35], distributed scheduling algorithms to schedule tasks with deadlines and resource requirements were proposed. When a task arrives at a node, the node determines the nodes to which the task is assigned to complete execution before its deadline. In doing so, the focused addressing algorithm was proposed where the task is scheduled on a node whose computing capability is estimated to be sufficient to complete the task before its deadline. This distributed scheduling was shown to be effective even in a hard real-time environment. In [36], scheduling algorithms for deadline-constrained jobs were covered in a multiprocessor environment. When a set of tasks is scheduled on a multiprocessor system to satisfy deadlines, a heuristic algorithm to consider the tasks with shortest deadlines was shown to effective for dynamic scheduling in real-time systems, where the task was represented by the worst case execution time, deadline, and resources requirement.

Our Contribution

An experimental data delivery job between the tier 1 and the tier 2 sites to satisfy a specific deadline should be scheduled and prioritized as a special data flow. To enhance the network performance guarantee provided for deadline constrained jobs motivated by the TeraPaths project, every job is scheduled with DLT (divisible load theory) and EDF (earliest deadline first), where each job is sorted in the order of the earliest deadline to be evaluated if each job is schedulable within its deadline. Real-time scheduling with divisible load theory is applicable to jobs with deadlines for heterogeneous Grid/cluster systems.

In this paper, real-time modeling from the perspective of divisible load theory is proposed for a heterogeneous Grid/cluster computing. In doing so, the minimum number of nodes to satisfy the deadline of a job are obtained through a homogeneous model transformed from a original heterogeneous model. Application specific scheduling adapted to the characteristics of a job is proposed, where the sequence of load distribution can be varied so that a job is communication-centric or computation intensive.

5.1 Models and problem description

5.1.1 Models and definition

In this section, the task, system, and scheduling model are discussed.

• **Task (Job) Model** A task T_i is assumed as an aperiodic independent task which is represented as (A_i, σ_i, D_i), where A_i is the arrival time of the task, σ_i is the data size of the task, and D_i is the duration of the task. The actual deadline is greater than or equal to A_i+D_i which is applied to a test to check if a task is schedulable.

• **System Model** A heterogeneous model in which processor capability and link bandwidth are different from one another is considered. A heterogeneous single level tree network is assumed in which the root processor, P_0 is connected to m number of children processors, P_1, P_1, ..., P_1 via LAN or WAN.

• **Divisible load scheduling model** In this heterogeneous single level tree network, the load scheduling policy follows the sequential distribution and staggered start policy from the DLT (divisible load theory) point of view [13]. The root processor sequentially distributes the fractions of a job to a number of children processors selected to be large enough to satisfy its deadline. The children processors participate in processing a job after receiving the entire load completely. The root processor is assumed not to participate in the job processing.

- **Real-time scheduling model** In 1973 Liu and Layland [37] proposed an preemptive earliest-deadline-first (EDF) scheduling and rate monotonic scheduling (RMS) policy for systems of independent aperiodic and periodic tasks with the relative deadline of each task. Especially, the processor utilization was proven to be maximized by dynamically assigning the priorities to the periodic tasks on the basis of the current deadlines. This utilization bound provides a simple and effective EDF schedulability test. Here, we select EDF policy for an independent aperiodic task (job) to handle the parameters, such as timing constraints, resource (processors and links) allocation requirements, and dynamic priority based on the deadlines. Whenever a new task arrives, the task is inserted into a queue of ready tasks on the basis of deadlines.

- **Definition**

α_i: The load fraction assigned to i^{th} processor.

w_i: The inverse computing speed on the i^{th} processor.

z_i: The inverse link speed of the i^{th} link.

T_{cp}: Computing intensity constant:

The entire load can be processed in $w_i T_{cp}$ seconds on the i^{th} processor.

T_{cm}: Communication intensity constant:

The entire load can be transmitted in $z_i T_{cm}$ seconds over the i^{th} link.

$T_{f,m}$: The finish time: Time when m number of of the processors complete computation.

5.1.2 Problem description

In the TeraPaths project [38] in Brookhaven National Laboratory a performance guarantee is considered for the problem of supporting reliable, large bulk data movement in terms of network bandwidth. A data delivery job to satisfy a specific deadline should be scheduled and prioritized as a special data flow. Moreover, if a job needs to be processed by a specific deadline, the problem to deal with this real-time scheduling is more challenging. In doing so, every job is sorted in the order of the earliest deadline to be evaluated if each job is schedulable within its deadline.

Divisible load theory can be extended to real-time scheduling of jobs with deadlines for homogeneous cluster systems [31]. However, heterogeneous models have not only been a major trend in network based Grid computing systems, but also is common in cluster computing systems. However, it is difficult to obtain the minimum number of nodes enough to satisfy a deadline of each job in a heterogeneous model from DLT model, since the equation for the finish time of a heterogeneous model is not simplified with regard to the number of nodes and the minimum number of nodes needed to satisfy the deadline of a job can not be obtained in a heterogeneous model.

To solve this problem, we propose a deadline-constrained network partitioning and application specific scheduling heuristics. In a deadline-constrained network partitioning algorithm, the minimum number of processors is obtained through transforming a heterogeneous model into a couple of homogeneous models. When the minimum number of nodes is applied into the heterogeneous model, the rejection ratio of the heterogeneous model can be found. The rejection rate of a heterogeneous model is studied in terms of a homogeneous model and the minimum number of nodes is obtained. In application specific scheduling, a heterogeneous model can be arranged according to the characteristics of the application. The jobs with EDF (earliest deadline first) policy are applied to the arranged (sorted) heterogeneous models, bases on computing intensive, communication centric, and both computing and communication centered applications.

5.2 Deadline-constrained scheduling in a heterogeneous model

5.2.1 Application specific scheduling in a heterogeneous model

According to the characteristics of a job, three kinds of load distribution sequences for the children processors to receive a fraction of a job from the root processor in a heterogeneous model can be considered. The sequence of load distribution affects a job execution time in a heterogeneous model as illustrated in Fig 5-2. The sequence of the children processors can be sorted for the purpose of load distribution in the decreasing order of computing speed, $\frac{1}{w_i}$, link speed, $\frac{1}{z_i}$, and a combination of computing speed,

$\frac{1}{w_i}$, and link speed, $\frac{1}{w_i}$. If a job is assumed to be computation intensive, computing speed based sequencing can be considered, or if it is communication centric, link speed based sequencing can be considered. A model considering both computing and link speed is also sometimes possible.

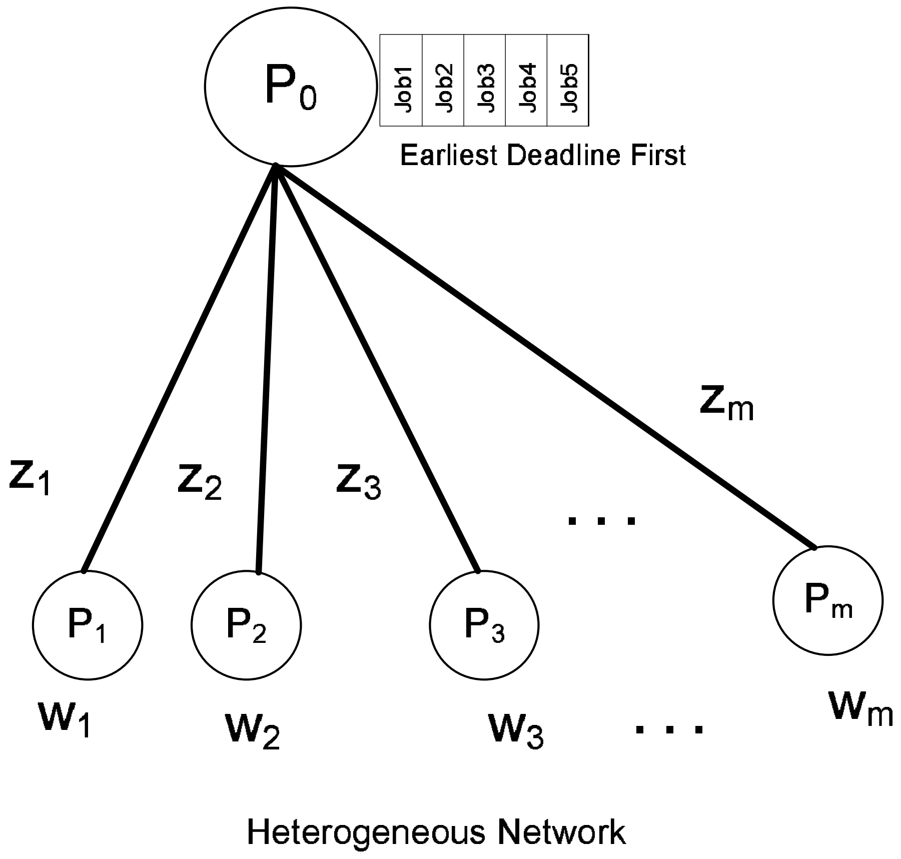


Figure 5-2: A heterogeneous model with deadline-constrained jobs.

• *HT-COMM* is applied to the heterogeneous network in which the order to receive load is sorted in the decreasing order of communication speed.

• *HT-COMP* is applied to the heterogeneous network in which the order to receive load is sorted in the decreasing order of computing speed.

• *HT-BOTH* is applied to the heterogeneous network in which the order to receive load is sorted in the decreasing order of the sum of communication and computing speed for a given processor.

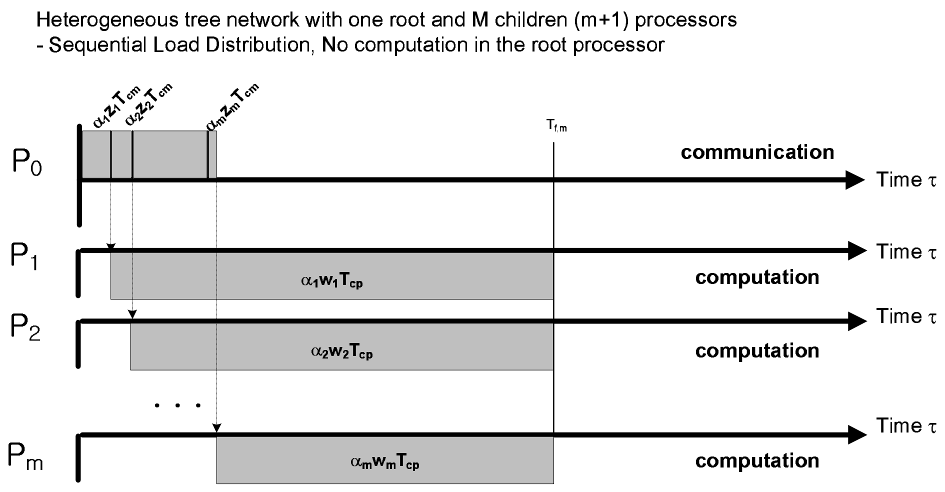

Heterogeneous tree network with one root and M children (m+1) processors
- Sequential Load Distribution, No computation in the root processor
α1z1Tcm
α2z2Tcm
αmzmTcm
Tf,m
P0
communication
Time τ
P1
Time τ
α1w1Tcp
computation
P2
Time τ
α2w2Tcp
computation
Pm
Time τ
αmwmTcp
computation

Figure 5-3: Timing diagram for a heterogeneous model.

From the timing diagram in Fig. 5-3, the equations for the finish time in a heterogeneous model are obtained as follows. We assume that one root processor and m number of children processors participate in a job processing

$$\begin{aligned} T_{f,m} &= \alpha_1 z_1 T_{cm} + \alpha_1 w_1 T_{cp} \\ &= \alpha_1 z_1 T_{cm} + \alpha_2 z_2 T_{cm} + \alpha_2 w_2 T_{cp} \\ &= \alpha_1 z_1 T_{cm} + \alpha_2 z_2 T_{cm} + \alpha_3 z_3 T_{cm} + \alpha_3 w_3 T_{cp} \\ &\quad \ldots \\ &= \alpha_1 z_1 T_{cm} + \alpha_2 z_2 T_{cm} + \ldots + \alpha_m z_m T_{cm} + \alpha_m w_m T_{cp} \end{aligned} \tag{5.1}$$

From equation (5.1),

$$\alpha_i = \frac{w_{i-1}T_{cp}}{w_iT_{cp} + z_iT_{cm}}\alpha_{i-1} = q_i\alpha_{i-1} \tag{5.2}$$

$$where \quad q_i = \frac{w_{i-1}T_{cp}}{w_iT_{cp} + z_iT_{cm}}, \qquad i = 2, 3, \ldots m$$

All of the load fractions is summed up 1 in the following equation (5.4).

$$\alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_m = 1 \tag{5.3}$$

$$\alpha_i = q_i\alpha_{i-1} = (\prod_{l=2}^{i} q_l)\alpha_1, \qquad i = 2, 3, \ldots m \tag{5.4}$$

Plug-in equation (5.3) into (5.4), we obtain

$$\begin{aligned} \alpha_1 + q_2\alpha_1 + (q_3q_2)\alpha_1 + \ldots + (\prod_{l=2}^{m} q_l)\alpha_1 &= 1 \\ \left[1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)\right]\alpha_1 &= 1 \\ \alpha_1 &= \frac{1}{\left[1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)\right]} \end{aligned} \tag{5.5}$$

The finish time for the job assigned to m children processors, $T_{f,m}$ is achieved as

$$\begin{aligned} T_{f,m} &= (z_1T_{cm} + w_1T_{cp})\alpha_1 \\ &= \frac{z_1T_{cm} + w_1T_{cp}}{\left[1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)\right]} \end{aligned} \tag{5.6}$$

5.2.2 Deadline-constrained network partitioning

Transforming a heterogeneous network into a homogeneous one

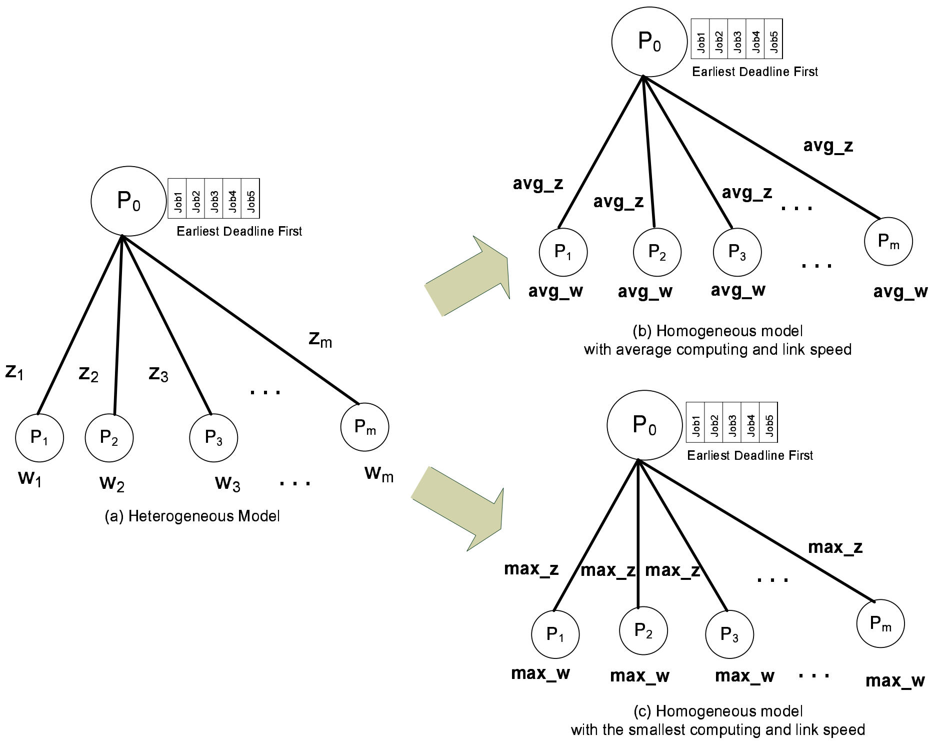


Figure 5-4: Transformation of a heterogeneous network into a homogeneous network

we examine two possibilities to transform a heterogeneous network into a homogeneous one. One is a homogeneous network using average computing ($\frac{1}{avg\_w}$) and average link speed ($\frac{1}{avg\_z}$) in Fig 5-4 (b) and the other one is a homogeneous network using the slowest computing ($\frac{1}{max\_w}$) and slowest link ($\frac{1}{max\_z}$) speed of a heterogeneous network in Fig 5-4 (c). The reason to transform a heterogeneous model into a homogeneous model is that the minimum number of processors to satisfy the deadline of each job for a homogeneous network is used in evaluating the original heterogeneous network. This minimum number of processors for a job can be obtained only from a homogeneous model with equal computing and link speed. This minimum number of processors is again applied to the original heterogeneous network to test if a job with a deadline constraint is schedulable. This is our schedulability test. The same minimum number of children processors from the heterogeneous network is selected and applied to the solution for the finish time in a heterogeneous model.

The inverse value of computing speed, w_i and and the inverse of link speed, z_i in a heterogeneous network is mapped to w and z in a homogeneous network respectively. All of the computing and link speed of a transformed homogeneous network are identical one another. There are two alternatives to decide the computing and link speed when a heterogeneous network is transformed into a homogeneous network. One is to select the slowest speed, and the other is to select the average speed among them. If the slowest speed is selected, the capability of computing and communication in a homogeneous network is always less than those in a heterogeneous network. Selection of the slowest speed in a heterogeneous network is regarded as a very conservative policy in terms of deadline constraints, since any combination of processor and link speed in a heterogeneous network is greater than those in a transformed homogeneous network. Alternatively, if the average speed is selected for a homogeneous network, the capability of computing and communication in a homogeneous network depends on how the processors and links are related to those in a heterogeneous network.

• **Minimum number of nodes from a homogeneous model**

Given the finish time for each job in (5.8), we can calculate the minimum number of nodes enough to satisfy the deadline of a task, as illustrated in [31]. For the transformed homogeneous model, it is assumed that the inverse computing and link speed are w and z, which can be the average value or the maximum value from the original heterogeneous model.

The finish time for the transformed homogeneous model is

$$T_{f,m} = (zT_{cm} + wT_{cp})\alpha_1 = (zT_{cm} + wT_{cp})\frac{1-q}{1-q^m} \tag{5.7}$$

$$where \qquad q = \frac{wT_{cp}}{zT_{cm} + wT_{cp}}$$

It is assumed that each task T = (A, σ, D) has a start time s, then we have constraints that the sum of the finish time and the start time should be less or equal to the sum of the arrival time and duration of a task.

$$T_{f,m}\sigma + s \leq A + D \tag{5.8}$$

$$(zT_{cm} + wT_{cp})\frac{1-q}{1-q^m}\sigma \leq A + D - s \tag{5.9}$$

Through simplification, we have the following relationship

$$q^m \leq 1 - \frac{zT_{cm}\sigma}{A + D - s} \tag{5.10}$$

$$m \ln q \leq \ln \gamma, where \qquad \gamma = 1 - \frac{zT_{cm}\sigma}{A + D - s} \tag{5.11}$$

Finally, the minimum number of nodes for a transformed homogeneous model, m can be obtained as (5.12), since the value, q is between 0 and 1, $\ln q$ is negative.

$$m \geq \lceil \frac{\ln \gamma}{\ln q} \rceil \tag{5.12}$$

- **Two-level schedulability test for a heterogeneous network**

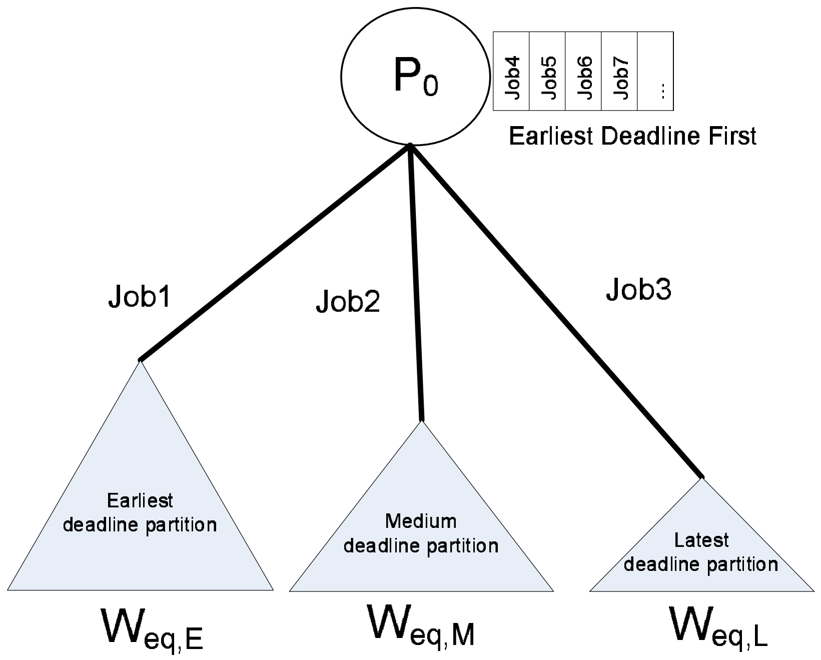
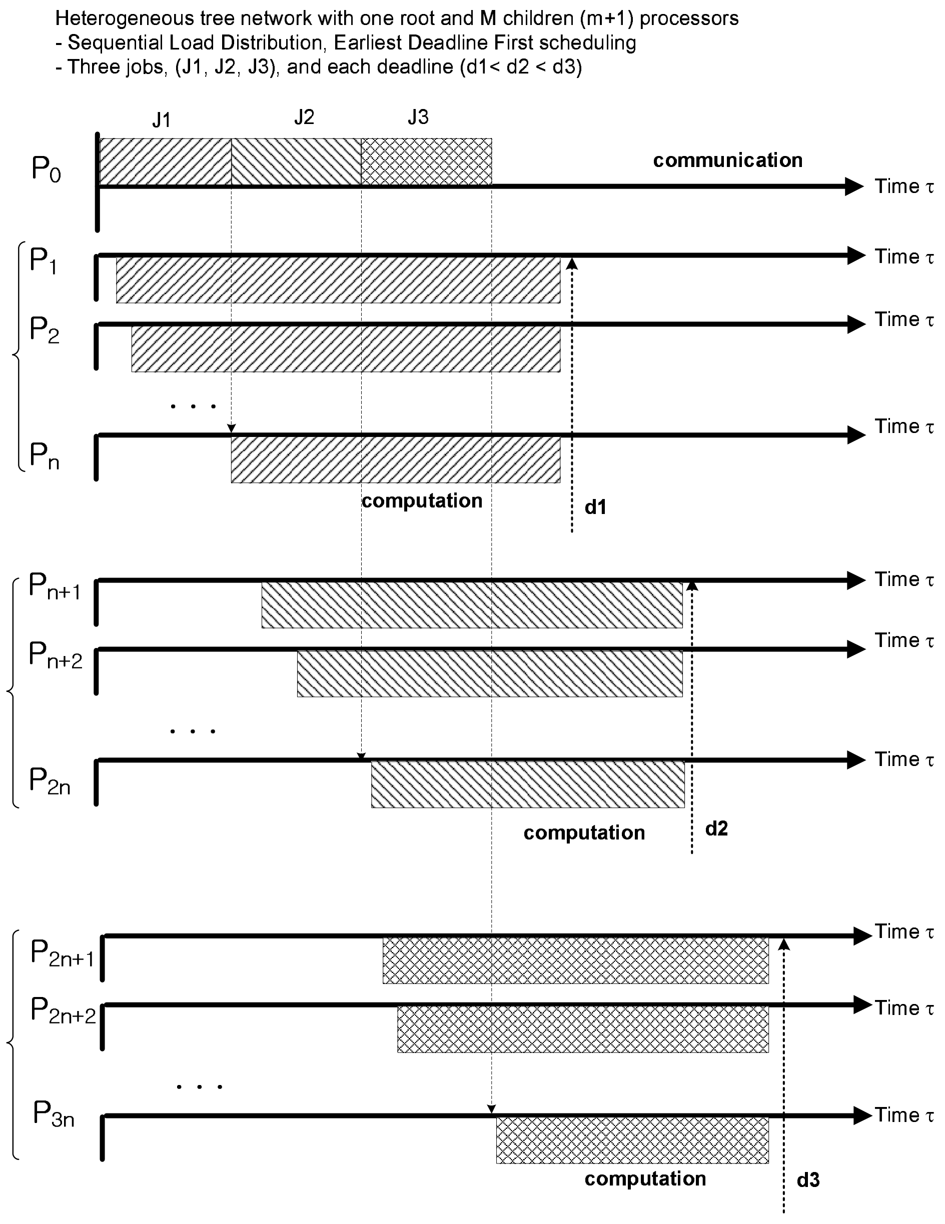


Figure 5-5: Deadline-constrained network partitioning in a heterogeneous tree network

The entire system or network is fragmented into several smaller networks dedicated to a job assigned by the root processor. The number of the selected children processors are the same as the minimum number of nodes obtained from a transformed homogeneous network. As illustrated in Fig 5-5, a job with the earliest deadline is assigned to a minimum number of children processor of high capability in the first sequence and called a highest priority partition. A job with the next deadline is assigned to the next minimum number of children processors from the remaining nodes and called medium priority partition, and there is a lowest priority partition respectively. Of course the use of more than three priority fragments could be contemplated.

As illustrated in Fig 5-6, a job with the earliest deadline is assigned to children processor which have higher capability in the computing speed or link speed sorted



Heterogeneous tree network with one root and M children (m+1) processors
- Sequential Load Distribution, Earliest Deadline First scheduling
- Three jobs, (J1, J2, J3), and each deadline (d1< d2 < d3)
J1
J2
J3
P0
communication
Time τ
P1
P2
. . .
Pn
computation
d1
Pn+1
Pn+2
. . .
P2n
computation
d2
P2n+1
P2n+2
. . .
P3n
computation
d3

Figure 5-6: Timing diagram for deadline-constrained network partitioning in a heterogeneous tree network.

according to the order of load distribution. Within a partition, a fraction of job is sequentially distributed to the children processors selected. Therefore, we can obtain the execution time of each job assigned to children processors, and evaluate whether or not each job is schedulable enough to satisfy its deadline. The goal is to schedule more jobs to satisfy their deadline from the perspective of the entire network.

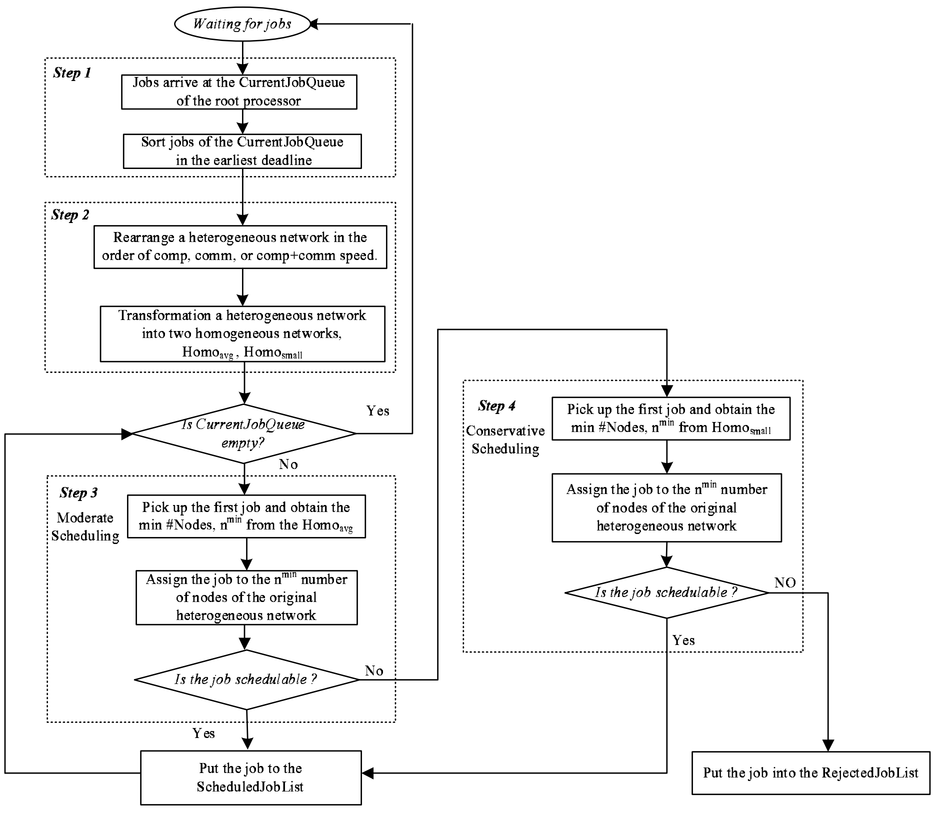

Waiting for jobs
Step 1
Jobs arrive at the CurrentJobQueue of the root processor
Sort jobs of the CurrentJobQueue in the earliest deadline
Step 2
Rearrange a heterogeneous network in the order of comp, comm, or comp+comm speed.
Transformation a heterogeneous network into two homogeneous networks, Homo_avg, Homo_small
Is CurrentJobQueue empty?
Yes
No
Step 3
Moderate Scheduling
Pick up the first job and obtain the min #Nodes, n^min from the Homo_avg
Assign the job to the n^min number of nodes of the original heterogeneous network
Is the job schedulable ?
No
Yes
Step 4
Conservative Scheduling
Pick up the first job and obtain the min #Nodes, n^min from Homo_small
Assign the job to the n^min number of nodes of the original heterogeneous network
Is the job schedulable ?
NO
Yes
Put the job to the ScheduledJobList
Put the job into the RejectedJobList

Figure 5-7: Two-level schedulability test in a heterogeneous model.

In Fig 5-7, a two-level schedulability test algorithm is proposed. In the moderate scheduling test, the minimum number of nodes are obtained from the homogeneous network , HM_{avg} transformed by the average computing and link speed in a original heterogeneous network, and applied to the schedulability test to verify the assigned job. Then in conservative scheduling test, the minimum number of nodes are obtained from the homogenous model, HM_{slow} with the slowest computing and link speed in a original heterogeneous network. The only minimum number of nodes are assigned to a job and used in conservative test if the job is schedulable. The reason to select the slowest processor from a heterogenous network is that the minimum number of nodes to satisfy the deadline of a job in a homogeneous network must satisfy the deadline of the job in a heterogeneous network only if every processor and link speed in a heterogeneous is larger than that in a homogeneous network.

Algorithm 3 Deadline-constrained scheduling algorithm

```
1: • Step 1 : Jobs with deadline arrive at job queue in the root node and sorted in the order of the earliest deadline first.
2: • Step 2 : Transformation of a heterogeneous to homogeneous model
3: Step 2-1 : Rearrange a heterogeneous model in the order of decreasing computing speed, link speed, or both of computing and link speed.
4: Step 2-2 : Transform either average or smallest computing, and link speed in a heterogeneous model into a homogeneous model, $HM_{avg}$ or $HM_{slow}$.
5: • Step 3 : Moderate Schedulability test
6: Step 3-1 : Obtain the minimum number of nodes, $n^{min}$ to satisfy deadline of the selected job in a homogeneous network, $HM_{avg}$
7: Step 3-2 : Obtain the finish time, $T_{f,n^{min}}$ with $n^{min}$ for the original heterogeneous network.
8: Step 3-3 : If $T_{f,n^{min}}$ + start time, $s <=$ Deadline, then accept the job, else go to step 4.
9: • Step 4 : Conservative Schedulability test
10: Step 4-1 : Obtain the minimum number of nodes, $n^{min}$ to satisfy deadline of the selected job in a homogeneous network, $HM_{slow}$
11: Step 4-2 : Obtain the finish time, $T_{f,n^{min}}$ with $n^{min}$ for the original heterogeneous network.
12: Step 4-3 : If $T_{f,n^{min}}$ + start time, $s <=$ Deadline, then accept the job, else reject it.
13: • Step 5 : Count the accepted jobs to satisfy deadline
```

5.3 Performance analysis and comparison

5.3.1 Simulation parameters

In this experiment, a fully heterogenous network is considered to be applicable to more general Grid systems and cluster systems. It is assumed that the root processor is fully connected to all of the children nodes.

The configuration of the simulation follows most of the previous simulation model explained in [31]. That is, for a task (job), $T_i = (A_i, \sigma_i, D_i)$, inter-arrival times, A_i are assumed to follow the exponential distribution with a mean of $\frac{1}{\lambda}$, and average data variation, σ_i, are the normal distribution, and the durations, D_i follow the uniform distribution. The system load is a metric defined as

$$SystemLoad = \frac{TotalTaskNumber \times ExecutionTime}{TotalsimulationTime}.$$

To evaluate the performance of real-time scheduling, the metric, *RejectionRatio* is used, which is the ratio of the number of jobs rejected to the total number of jobs arriving the root processor.

The another metric, *FinishTime* is the time each job is completed with the number of processors in a respective network from DLT's point of view.

5.3.2 Performance analysis

It is meaningful to effectively schedule jobs with deadlines in a heterogeneous model, since a heterogeneous model has been dominant on a Grid computing as well as cluster computing. Through the DLT methodology we can schedule more deadline constrained real time jobs to satisfy their deadlines. The inverse value of computing speed, w_i is randomly assigned within 1 to 100 and the inverse value of link speed, z_i is also randomly assigned within 0.1 to 1.0.

System Model	Computing / Link Speed	Scheduling Policy	# Nodes
HM (Homo)	MIN AVG MAX	EDF FIFO	AN (All Nodes) MN (Min Nodes)
HT (Hetero)	RANDOM COMP ($\frac{1}{w_i}$) COMM ($\frac{1}{z_i}$) BOTH ($\frac{1}{w_i} + \frac{1}{z_i}$)	EDF FIFO	AN (All Nodes) MN (Min Nodes)

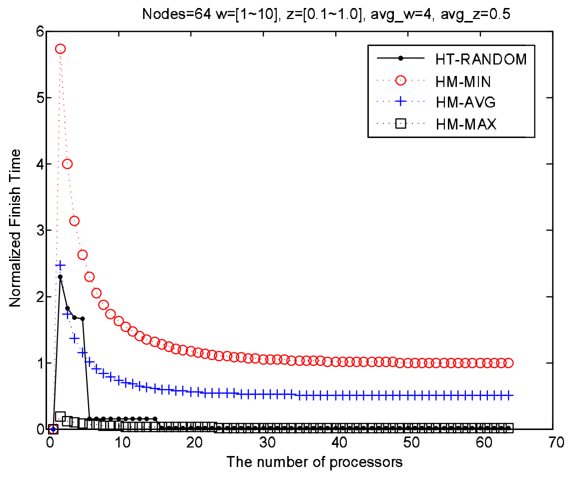
Table 5.1: The notations for the algorithms tested in the simulation.

In the TABLE 5.1, for the model, HM is an abbreviation for homogeneous and HT for heterogeneous. For a homogeneous model, computing/link component is used to obtain the minimum number of nodes in satisfying deadline of a job. MIN is a homogeneous model with the minimum computing and link speed transformed from the heterogeneous model, AVG is with the average speed, and MAX represents the maximum speed case. For a heterogeneous model, the nodes in a heterogeneous model are sorted according to the characteristics of application: $COMP$ is sorted in the decreasing order of computing speed, $COMM$ is communication speed, $BOTH$ is a combination of computing and communication speed, and the $RANDOM$ is the randomly generated nodes. In scheduling policy, EDF is the earliest deadline first algorithm, and FIFO is First-In-First-Out policy. For the number of nodes used in job execution, MN means that the minimum number of nodes are selected from the available nodes of the systems to process a job, and AN means that all of the nodes in the system are used. These four components can be combined to evaluate the algorithms. For example, HT-EDF-MN means that the minimum number of nodes are selected in scheduling jobs with the earliest deadline first (EDF) policy in a heterogeneous model.

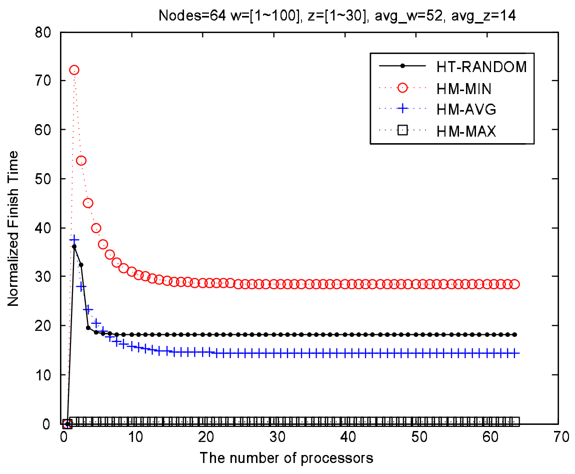
• **Finish time** The finish times are obtained for the original heterogeneous model and various transformed homogeneous models. As shown in Fig. 5-8 (a), the homogeneous model with the slowest computing and link speed, HM-MIN has the largest finish time, and for the heterogeneous model, HT-RANDOM has good performance similar to the homogeneous model with the fastest computing and link speed, HM-MAX as the number of processors increase when the ratio of link to computing speed, $\sigma = \frac{avg\_zT_{cm}}{avg\_wT_{cp}}$, is small as '0.125'. Interestingly, when the ratio σ becomes larger, '0.27' in Fig. 5-8 (b), $HT - RANDOM$ is a little bit larger than $HT - AVG$ in the finish time. We select the model that the ratio σ is relatively large in the following simulations, since the parameters with the large σ are easily compared with the homogeneous model in earlier work [31].

• **Effect of the System Load** As the rejection ratio of jobs is illustrated in Fig 5-9, the rejection ratio for the heterogeneous network is between the average homogeneous network and minimum homogenous network. In several simulations, the rejection ratio of the heterogeneous model is varied on the computing and link speed randomly generated. As the system load increases, the rejection ratio also is increased, since the overhead is imposed on each model.

• **Effect of Inter-arrival time**



(a) Fast computing and link speed



(b) Slow computing and link speed

Figure 5-8: The finish time for heterogeneous and transformed homogeneous models.

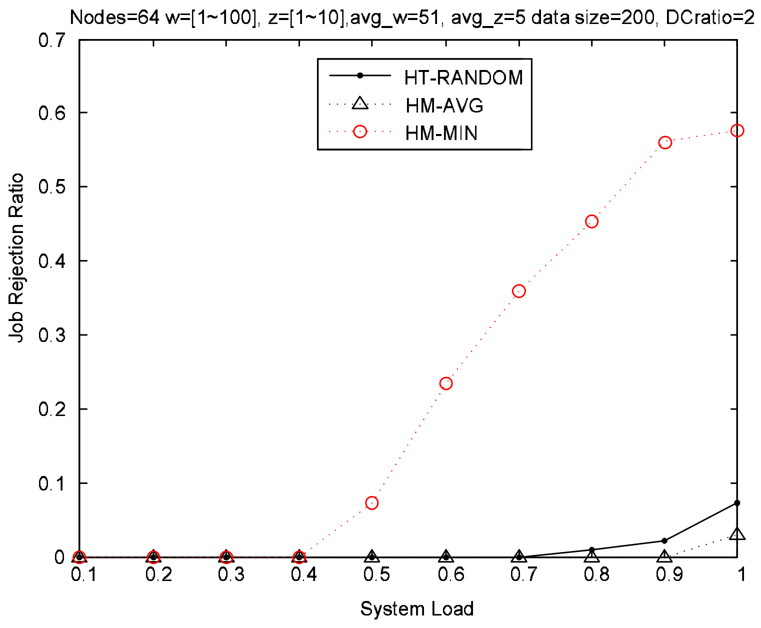


Figure 5-9: The rejection ratio for heterogeneous and transformed homogeneous models with the system load.

Since inter-arrival time is modeled to exponentially be distributed, as the inter-arrival time increase, the less jobs arrive. As the less jobs arrive, the rejection ratios of three models, HT-RANDOM, HM-AVG, and HT-BOTH are also decreased to zero after inter-arrival time is above 1200 as shown in Fig 5-10. However, the rejection rate of HM-MIN keeps higher than other models, such as HT-RANDOM or HM-AVG, since the finish time of HM-MIN is too large to be accepted. The reason the finish time of HM-MIN is large is that the model HM-MIN consists of the slowest computing and link speed from the original heterogeneous model.

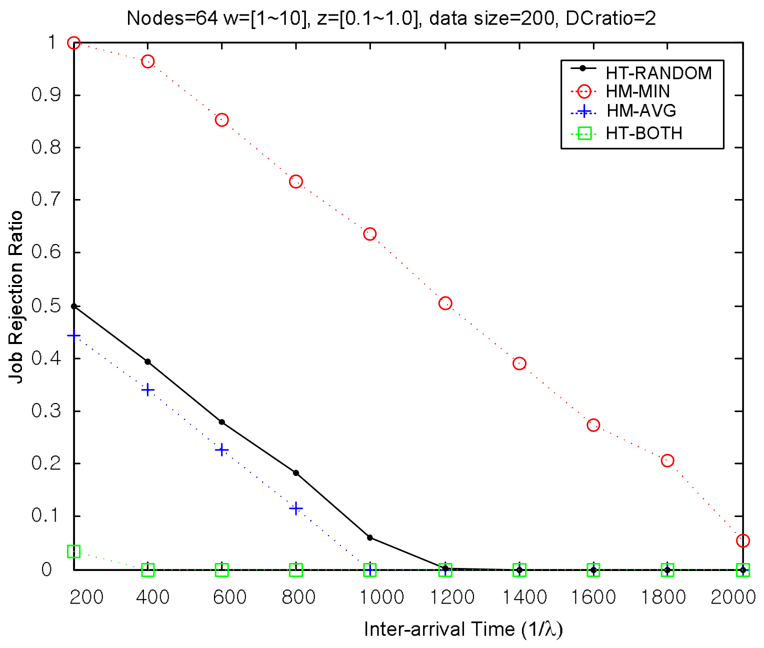
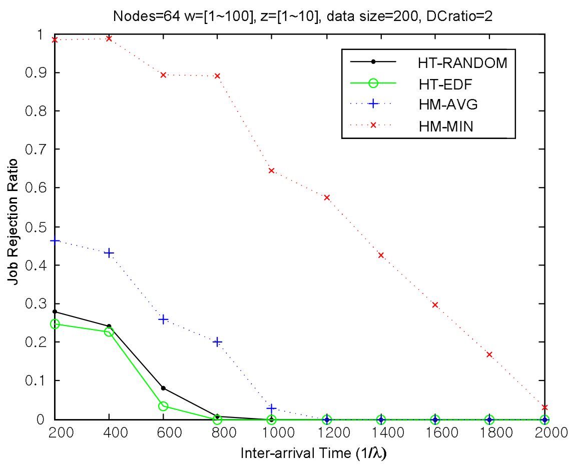


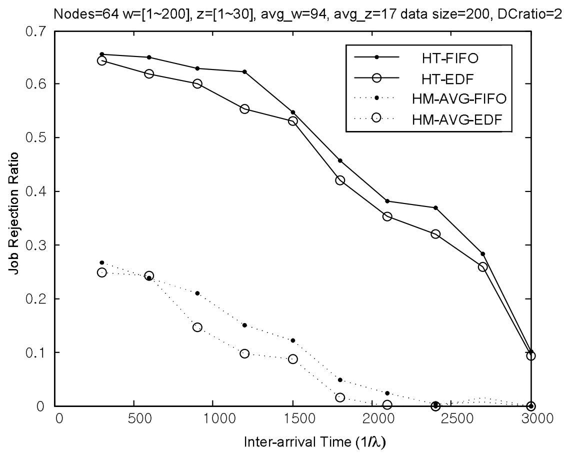
Figure 5-10: The rejection ratio for heterogeneous and transformed homogeneous models with inter-arrival time $(1/\lambda)$.

- **EDF vs. FIFO**

In Fig 5-11, the effect of the order of jobs, EDF and FIFO is investigated for either a heterogeneous or homogenous model. When the link speed with large $\frac{1}{z_i}$ is fast, the curve drops down very quickly as the inter-arrival time of jobs increases larger, as shown in Fig 5-11 (a), which means that the rejection rate is more sensitive to the number of jobs within a specific period. The difference of EDF and FIFO policies is not very large in the rejection rate, when the link speed is relatively fast. As for the case the link speed is slow in Fig 5-11 (b), it is clear that EDF policy shows better performance than FIFO policy, since the curve of EDF is below that of FIFO gradually. The rejection rate of a heterogeneous model, HT-FIFO and HT-EDF is higher than that of a homogeneous model, HM-AVG-FIFO and HM-AVG-EDF with average computing and link speed. The reason is that the finish time of a homogeneous model generally smaller than that of a heterogeneous model, when the link speed is relatively slow as shown in Fig 5-11 (b).



(a) Fast link speed (z_i is small)



(b) Slow link speed (z_i is large)

Figure 5-11: EDF versus FIFO in a heterogeneous and homogeneous model.

- **Application specific arrangement**

In a heterogeneous model, we can arrange the jobs to be adapted to application specific model. That is, according to the characteristics of application, the nodes participating in the system can be ordered as a computation intensive model, a communication centric model, and an integrated computation and communication model. The integrated model, HT-BOTH, shows the lowest rejection rate, while more jobs in communication centric model,HT-COMM are schedulable than those in computing intensive model, HT-COMP as shown in Fig 5-12. Interestingly, computing intensive model, HT-COMP, is likely to show high rejection ratio, and often higher than randomly ordered model, HT-RANDOM. Since the finish time is affected by communication speed and computing speed, in the randomly generated heterogeneous model, it can not be guaranteed that the jobs sorted in the order of computing speed, HT-COMP, are more admitted than the randomly generated model, HT-RANDOM.

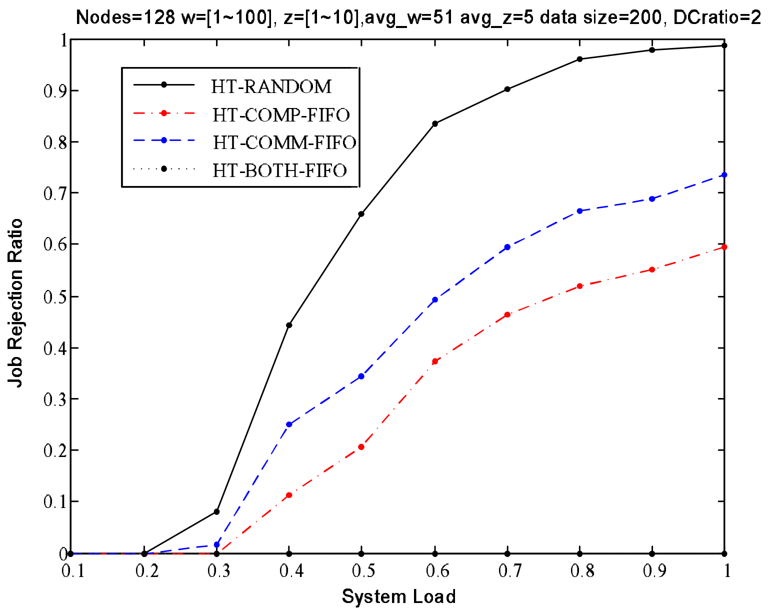
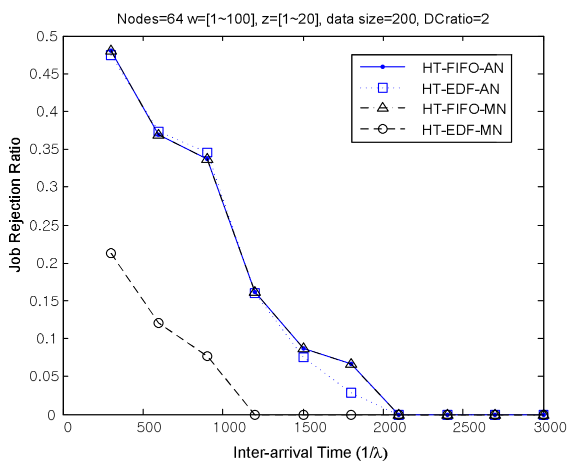


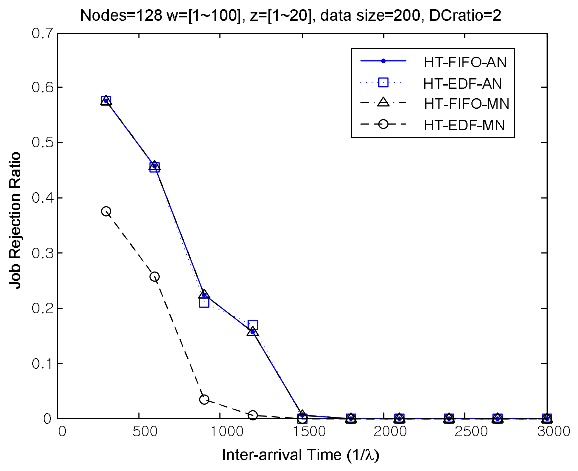
Figure 5-12: The rejection ratio of application specific heterogeneous systems.

• Effect of Network partitioning

Under the situation needed small nodes for the task, network partitioning methods using the minimum number of nodes for each task shows lower rejection ratio than



(a) Nodes = 64



(b) Nodes = 128

Figure 5-13: Minimum nodes network partitioning in a heterogeneous model.

those methods all of nodes involved. However, as shown in Fig 5-13, the greatest factor influencing the rejection rate is which scheduling policy is used. In the general case jobs are sorted in the order of the earliest deadline with the EDF policy performs better than the policy when the jobs are sorted in the order of arrival with the FIFO policy.

5.4 Conclusion

In this chapter, divisible load theory is extended to model deadline-constrained jobs with the EDF (earliest deadline first) policy in a heterogeneous Grid/cluster systems. According to deadlines, the entire network is fragmented into a couple of smaller networks with the number of nodes enough to satisfy each job's deadline. The heterogeneous network can be arranged to fit the characteristics of applications which are computing intensive, communication-centric, or both computing and communication.

Through a two-level schedulability test algorithms to test if each job completes before its deadline, we can check schedulability of a job under moderate condition or strict condition for heterogeneous systems. Our proposed real-time scheduling algorithms are not only one of feasible solutions for a heterogeneous model common in the real computing and network environment, but shows also good performance than in terms of the rejection ratio of jobs.

Chapter 6

Future Research and Conclusion

6.1 Conclusion

In this dissertation we proposed sophisticated scheduling algorithms dealing with concurrent communication, communication interference on computation, workloads from multiple sources, and deadline constrained jobs in large scale heterogeneous networks and computational Grids. For the first two problems, concurrent communication and communication interference on computation, concurrent scheduling and interference aware scheduling algorithms are proposed and the closed-form solutions for the execution time are mathematically developed and analyzed via DLT (divisible load theory). Specifically concurrent scheduling algorithms are applied in parallel video processing application to show better performance than the previous algorithms. It is very meaningful to consider the effect of communication interference one of the affecting factors on system performance and develop the closed-form equation for the execution time in scheduling algorithms, since interference, to the best of our knowledge, is not formulated in the closed-form solution in the design parameters to date. Also, concurrent scheduling and interference aware scheduling can be combined to apply to parallel video processing to show more realistic modeling.

For the next two problems, workloads from multiple sources and deadline constrained jobs, multi-source scheduling algorithms are devised and the solutions are experimentally obtained using a genetic algorithm and DLT and real-time scheduling with an integrated EDF (earliest deadline first) policy. This DLT is applied to fully heterogeneous Grid/cluster systems. Scheduling workloads originating from multiple sources is a combinatorial problems and heuristic algorithms and its near-optimal solutions are obtained via the approach of genetic algorithms, since the optimal solution can not be found in the closed-form formula. The result via the use of a genetic algorithm shows that the plots do converge which we conjecture can be regarded as a globally optimal value for the experimental parameters. In real-time scheduling, we are challenged to integrate one of traditional real-time scheduling algorithms, the EDF policy, and DLT's sequential distribution scenario and apply the scheduling algorithm to a fully heterogeneous model, since real-time scheduling and DLT is a very good match in terms of scheduling arbitrarily divisible workloads. Our proposed real-time scheduling with EDF and DLT applicable to fully heterogeneous model is believed to be meaningful in that DLT's scheduling algorithms and mathematical solutions are combined with algorithms in real-time scheduling for either periodic or aperiodic task. Hence this series of work in the dissertation can be applied to difficult problems in mission critical real-time applications

6.2 Future research

In the future, real-time scheduling proposed in this dissertation can be extended to deal with individual jobs originating from multiple sources under deadline constraints. Also, multi-round scheduling from DLT's perspective can be combined with RMS (rate monotonic scheduling) in the real-time scheduling for a periodic tasks.

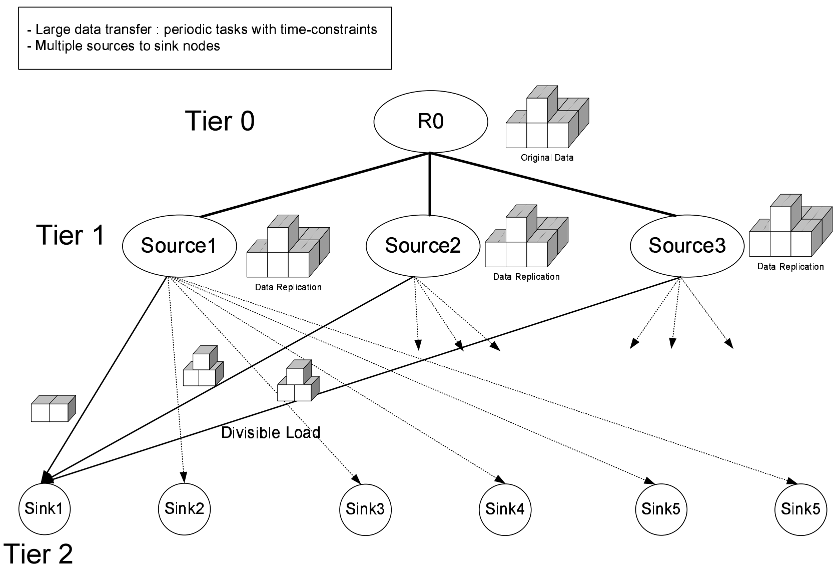


Figure 6-1: Multi-source real-time scheduling scenario

For example, as shown in Fig. 6-1, a periodic data transfer can occur among multi-source and sink sites over a guaranteed QoS network. Multiple source nodes are assumed to have data replication that can be divided into the number of fractions and distribute a fraction of data to sink nodes. Here, we can assigned a number of fractions according to the size of bandwidth assigned to the links. With DLT and RMS scheduling, we can schedule periodic massive data transfer with time-constraints over the deterministic bandwidth for data transfer. Through multi-source real-time scheduling algorithm, we can obtain better throughput of data transfer.

Bibliography

[1] D. Altilar and Y. Parker. Optimal scheduling algorithms for communication constrained parallel processing. *EuroPar 2002.*, 2002.

[2] Z. Xu K. Hwang. *Scalable Parallel Computing.* McGRAW-HILL, 1998. Chapter 6, pp.273-293.

[3] S. Yalamanchili J. Duato and L. Ni. *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann, 2003.

[4] http://www.netlib.org/benchmark/top500.

[5] A. A. Kassim P. Li, B. Veeravalli. Design and implementation of parallel video encoding strategies using divisible load analysis. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):1098–1112, 2005.

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

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

[8] Dimitrios Katramatos and Steve J. Chapin. A cost/benefit estimating service for mapping parallel applications on heterogeneous clusters. In *Proc. of the 2005 IEEE International Conference on Cluster Computing*, Boston, MA, USA, September 2005. IEEE.

[9] US Atlas Tier-2 sites list : http://www.usatlas.bnl.gov.

[10] the TeraPath Project : https://www.racf.bnl.gov/terapaths.

[11] H.J. Kim M. Suresh, S. N. Omkar. Parallel video processing using divisible load scheduling paradigm. *Korean Journal of Broadcast Engineering*, 10(1), 2005.

[12] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems.* IEEE Computer Society Press, 1996.

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

[14] 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.

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

[16] D. A. L. Piriyakumar and C. S. R. Murthy. Distributed computation for a hypercube network of sensor-driven processors with communication delays including setup time. *IEEE Transactions on Systems, Man, and Cybernetics-PART A: Systems and Humans*, 28(2):245–251, March 1998.

[17] J. T. Hung, H. J. Kim, and T. G. Robertazzi. Scalable scheduling in parallel processors. *2002 Conference on Information Sciences and Systems*, March 2002. Princeton University.

[18] L. Carter J. Ferrante B. Kreaseck, H. Casanova and S. Nandy. Interference aware scheduling. *International Journal of High Performance Computing Applications*, 20:45–59, February 2006.

[19] W. Glazek. A multistage load distribution strategy for three dimensional meshes. *in special issue of Cluster Computing on Divisible Load Scheduling*, 6(1):31–40, January 2003. Kluwer Academic Publishers.

[20] Cray Inc. Cray xd1, xt3, t90 technical summary.

[21] P. Pepeljugoski K. Stawiasz J. Trewhella D. Booth W. Nation C. DeCusatis D. M. Kuchta, J. Crow and A. Muszynski. Low cost 10 gigabit/s optical interconnects for parallel processing. In *The Fifth International Conference on Massively Parallel Processing Using Optical Interconnections*, Washington, DC, USA, June 1998. IEEE.

[22] J. A. Kash D. M. Kuchta M. B. Ritter A. F. Benner, M. Ignatowski. Exploitation of optical interconnects in future server architectures. *IBM Journal of Research and Development*, 49(4/5), 2005.

[23] 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.

[24] 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.

[25] A. Legrand O. Beaumont and Y. Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*. IEEE, 2003.

[26] 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.

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

[28] 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.

[29] T.G. Robertazzi and D. Yu. Multi-source grid scheduling for divisible loads. In *Proceedings of the 2006 Conference on Information Sciences and Systems*, Princeton University, Princeton, NJ, USA, March 2006.

[30] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 1998.

[31] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for cluster computing. In *Proc. of the 13th Real-Time and Embedded Technology and Application Symposium*, Bellevue, WA, USA, April 2007. IEEE.

[32] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling with different processor available times. In *The International Conference on Parallel Processing (ICPP)*, XiAn, China, September 2007. IEEE.

[33] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Enhanced real-time divisible load scheduling with different processor available times. In *The 14th IEEE International Conference on High Performance Computing (HiPC)*, Goa, India, December 2007. IEEE.

[34] D. Yu and T.G. Robertazzi. Divisible load scheduling for grid computing. In *Proc. of the IASTED International Conference on Paralle and Distributed Computing and Systems*, Los Angeles, USA, Nov 2003. IEEE.

[35] K. Ramamritham, J.A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123, August 1989.

[36] K. Ramamritham, J.A. Stankovic, and P.F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 01(2):184–194, 1990.

[37] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[38] D. Katramatos and D. Yu. The terapaths testbed:exploring end-to-end network qos. In *International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, Orlando, Florida, USA, May 2007. ICST.