CHAPTER ONE

INTRODUCTION

1.1 Background of the Study.

Computational and data intensive e-Science and e-Collaboration applications involve special class of scientific services or instrument (located across various organizations) that are geographically distributed. These resources could be computational systems (such as super computers, clusters, or even powerful ultra-high end engineering workstations), special class of devices (such as remote sensors) and even storage systems. A number of data and computational intensive applications need more computing power than can be offered by a single resource in order to solve problems within feasible/reasonable time and cost. The LAN/SWITCH connected clusters (of computers) platform has been employed to solve computationally intensive problems (Buyya. R, 1999) however, they alone cannot offer the computational power demanded by such applications. All these means that these geographically distributed resources need to be logically coupled together to make them work as a unified resource. This led to the popularization of a field called grid computing – i.e. grid computing is a computational technique to harness distributed resources as a unified process. Grids consist of the aggregation of numerous dispersed computational, storage and network resources, able to satisfy even the most demanding computing jobs (Pieter, T et.al, 2007). Grids using optical transport networks are commonly referred to as Lambda Grids.

E-science project like Nigeria Communication Commission (NCC) project requires several hundreds of data file on the initiative to connect/linkup all the federal universities with their teaching hospitals referencing “Nnamdi Azikiwe University link establishment to the Nnewi Teaching hospital”. This project could be referenced direct point link via fiber optics (Lambda connection) on 4mbps capacity and not a Grid infrastructure. Since other Nigeria Universities are not yet in connectivity with one another in other to showcase its full scale infrastructure of grid framework, in other to enhance research work, e-collaboration, result computation and school portal security academic architecture.

Many data-intensive e-science applications like electronic Very Long Baseline Interferometry (e-VLB (Wolfgang, S and Behrend, D, 2007) and Genomes – to – life (GTL) require aggregating several hundred Gigabytes of data files from distributed databases to computing resources (such as super computers) frequently in real time. Since data is aggregated at the time of computation, the time required to transfer the data over the network may be the main computational bottleneck. Even a single second of idle time, during which the data is being aggregated, may result in the loss of several teraflops of computational power. Therefore, minimizing the delay in data aggregation is the key to improving the overall system throughput (Shen, S. et.al., 2008) (Savera. T.W et.al., 2008). A reliable and dedicated infrastructure available on demand is a key resource for

data intensive e-science application. Lambda Grid, which are backbone networks supported on optical fiber technology can provide such an infrastructure – since it offers end – to – end optical circuit (also called wavelength or lambda – an optical connection established over a certain wavelength) between two end points (Shen, S. et.al.,2008).

However, a number of networking problems need to be solved before lambda grid can fully and optimally support such data intensive e-science applications (Lee, H., et.al., 2003). These include:

(i) Dynamic provisioning of end – to – end circuit i.e. Lambda.

(ii) Authentication, Authorization, Accounting (AAA) for user request.

(iii) Transport protocols in those end – to – end circuits for different types of applications, and

(iv) Algorithm to schedule circuits (Brown, M.D., 2003): notes that "Lambdas" are resources to be scheduled like any other computer or storage resource). It is established in the literature that on Lambda Grids different time scheduling and fair sharing schemes for data transmission will tremendously affect the aggregation finish time and throughput.

Resource scheduling algorithm for the lambda grid such as machine scheduling, have been studied extensively in the literature. Surveys of this topic can be found in (Garey, M.R. and David, S.J., 1979) and (Hall N.G and Sriskandarajah .C, 1996). Such problems

include single processor scheduling, multi-processor scheduling, and open-shop, flow - shop, and job-shop scheduling problems to name but a few. However, these are not applicable to the proposed problem setting of the dissertation since the resources (lambda) that is considered in the dissertation are not independent; rather, they have connectivity relationship among them. The closest problem setting as described in the literature considers scheduling file transfers over a network when file sizes and the maximum number of file transfers possible from each node are given (Coffman, E.G. et.al., 1985). This problem considers a fully connected mesh network, and hence, the algorithms described are not applicable to lambda grids, which have sparse connectivity.

Independently, the problem of reserving bandwidth in a lambda grid for a prescribed connectivity has been studied by many researchers as available literatures show. A bandwidth scheduling algorithm that computes the available time slots on a lambda grid between the source and destination has been studied in [Nageswara, S.V. et.al., 2005]. Same authors have proposed algorithms for computing the quickest paths, with a minimum end – to – end delays, to transfer a message for a given size from its source to a destination when bandwidth and delay constraints on the links are specified (Nageswara, S.V. et.al., 2004). In the literature, the virtual finish (ViFi) (Floyd. S and Nageswara, S.V, 2008) heuristic schedules file transfer over a shared path, depending on the earliest finish time for each file determined from a fair sharing scheme. A Varying Bandwidth List Scheduling (VBLS) heuristic to compute circuit over a lambda grid was studied in

(Vecrarghavan, M., et.al., 2003). Reports from the references (Buyya, R., 1999) (Pieter, T. et.al., 2007) indicate that the basic problem with most of these grid scheduling algorithms is that they assume an ideal communication system where all the resources are fully connected and communication between two resources can be used whenever needed. However, a few record studies (She, O et.al., 2007) (Banerjee, A. et.al., 2008) (Adami, D. et.al., 2006) have proposed the framework where the optical network has been considered as a resource in a similar way to a computing node, or storage resources and investigated the joint scheduling model for optical grid applications. (Banerjee,.A. et.al., 2008) Propose a hybrid approach that combines online and offline scheduling. Based on the transfer time of the file, the network resources are reserved.

Unfortunately, none of the above lambda grid scheduling algorithms dynamically readjusts the schedule to accommodate the actual amount of time that is required to transfer a file. Hence to minimize the delay in data aggregation in the lambda grid network, an adaptive network resource scheduling technique for the lambda grid is required. The lambda grid network resource scheduling technique has to dynamically readjust the scheduling of the optical light paths (optical circuits i.e. the lambdas) in order to re-provision idle lambdas to service urgent file transfer request from the grid connected nodes (super computers).

Consequently, it becomes imperative to embark on a dissertation focused on the investigation of an adaptive resource scheduling technique to minimize the delay in the data aggregation task required by the computational and data intensive e- science application running on the lambda grid network.

1.2 Statement of the Problem.

The problem of data aggregation delay in the lambda grid has enormous impact on the viability of certain e-science applications that have critical timing requirements. The loss of teraflops of super computing power as result of scheduling related delay has substantial impact on R & D (research and development) cost. This problem not only leads to delays in vital research breakthroughs for mankind but also, in some cases, outright project cancellation and wasted investment.

The lambda grid scheduling problem impacts on the accuracy and validity of e-science applications used in making vital forecasts related to natural disasters, evolving disease posture, climate change, issues concerning related to deep space exploration etc. The scheduling problem has far reaching consequences financially, politically and for security.

Though resource scheduling algorithms have been studied extensively in the literature, there are still shortcomings with these algorithms (Castillo. C. et.al., 2007)

The problem with virtual finish (ViFi) heuristic scheduler and the Varying Bandwidth List Scheduling (VBLS) heuristic (which computes varying bandwidth levels for different time ranges for a circuit over a lambda grid) (Veerarghavan, M. et.al., 2010) (Floy, .S. and Nageswara, S.V, 2008) is that they assume an ideal communication system where all the resources are fully connected and communication between two resources can be used whenever needed which hardly is the case (Buyya, R., 1999)(Pieter, T. et.al., 2008).

The Time – Path – Scheduling Problem (TPSP) technique (used to model the problem of aggregating data from distributed databases to one centralized super computer on lambda grid network) can not cover the situation where there might be data aggregation to more than one location simultaneously.

(Hall, N.G and Sriskandarajah, .C., 1996). The machine scheduling algorithm (modeled as single processor scheduling, multiprocessor scheduling, flow-shop, job-shop scheduling) are not well adapted to scheduling lambda grid resources since lambdas are not independent; rather, they have connectivity relationship among them.

Furthermore, none of the above lambda scheduling algorithms dynamically readjusts the schedule to accommodate the actual amount of time that is required to transfer large files.

Hence to minimize the delay in aggregation in the lambda grid network, an adaptive network resource scheduling technique for the lambda grid is required. The lambda grid network resource scheduler has to dynamically readjust the scheduling of the optical light paths (optical circuits i.e. the lambdas) in order to re-provision idle lambdas to service urgent file transfer request from the grid connected nodes (i.e. super computers).

This dissertation is focused on the improvement of an adaptive resource scheduling techniques to minimize the delay in the data aggregation task required by the computational and data intensive e-science application running on lambda grid networks. The proposed scheduling technique would minimize the blocking probability i.e. the probability of not scheduling a request within its window, minimizes fragmentation in the usage of each wavelength and maximizes network usage. The proof of concept is the development of an adaptive lambda grid resource scheduler to be tested and evaluated via simulation.

1.3 Aim and Objectives of the Study.

The aim of this dissertation is to develop an improved computational Lambda grid using Adaptive resource scheduling Technique in other to optimize the computational throughput. The aim would not be achieved without the actualization of these stipulated objectives:

(i) To develop an adaptive resource scheduling algorithm that minimizes file aggregation time for computational lambda grid.

(ii) Based on the algorithm developed, to develop and realize the algorithm in code using C++ programming language.

(iii) To create a digital model of a 24-node lambda grid as a case study lambda grid topology for evaluating the performance of the proposed scheduler using Packet tracer 7.1

(iv) To integrate and interface the codes with the analytical model topology of the 24-node lambda grid network using CORBA protocol.

(v) To carry out performance evaluation of the proposed system through comparative analysis with other existing scheduler on MATLAB simulation testbed.

1.4 Justification/Significant of Project.

It is well known that resources; computing resource, storage resources specialized devices (e.g. telescopes, sensors, etc) cannot be easily co-located: essentially being geographically dispersed. Consequently, there is the need to harness these geographically distributed resources to effect a computational task. The lambda grid is an answer to this need. However due to the shortcoming with existing lambda grid scheduling algorithm, a research project becomes necessarily to investigate the development of appropriate grid algorithm to overcome the limitation of existing scheduling algorithm.

With the emergence of e-collaboration and e-science application a research work that focuses on developing scheduling algorithm to fully harness the capability of existing lambda grid, making full exploitation of e-science application becomes significant.

This dissertation makes important contributions:

(i) An adaptive grid scheduling algorithm based on the use of the time –path scheduling problem abstraction.

(ii) A realization of the algorithm in (i) in code

The work should prove invaluable to researchers pursuing similar subjects.

1.5 Scope/ Limitation of the Study.

Though a comprehensive examination grid resource scheduling problem involves both network and computing resource scheduling, this dissertation is focused on the improvement of grid network resource scheduling specifically, the scope of this work is limited to the dynamic provisioning of light paths (i.e. lambda) such as to speed up data aggregation in lambda grid network.

CHAPTER TWO

LITERATURE REVIEW

2.1 Chapter Overview.

This chapter presents a review of the literature relevant to the research. It reviews literature on the emergence of the lambda grid and the force pushing the development of the lambda grid. Though the literature documents and comments from different authors as to the key reason(s) for the emergence of the lambda grid, a diligent study of available literature showed that grid computing emerged as a means of compiling together numerous heterogeneous and geographically distributed computational and storage resources to make them work as a unified resource.

Technical documentation relating to the modeling of grid scheduling problem is reviewed. Though various literatures on the subject in one way or the other attempt at describing the problem of resource scheduling for the lambda grid, this chapter noted that two things were evident in the various documents;

(i) The representation of Lambda grid network topology as graphs (mathematical modeling) and;

(ii) The formulation of the problem of aggregating large data files from distributed database in a Lambda grid network as a Time-path Scheduling Problem (TPSP).

(Coffman, E.C. et.al. 1985) (Taesombut, N. et.al.,2006) (Banerjee, A. et.al., 2006) confirm TPSP to be NP complete. However, as reported, TPSP as a problem is exceptionally hard as it cannot be solved within polynomial time. Consequently, the literature reported the proposition of heuristics for its solution. Hence this chapter presented literature on heuristics for TPSP and algorithms to determine route and schedule.

2.2 Emergence of the Lambda Grid

Some authors (Bunya, R. et.al., 2002) commenting on the emergence of grid computing expressed the view that grid computing is getting popular day by day with the emergence of the internet as a media and the wide spread availability of powerful computers and networks as low-cost commodity components. Through diligent study of available literature, it can be summarized that grid computing emerged as a means of coupling together numerous heterogeneous and geographically distributed computational and storage resources to make them work as a unified resource.

By coupling numerous heterogeneous computational and storage resources distributed over various locations, Grids are able to satisfy the ever increasing demand of both processing and storage power, surpassing the capabilities of each of its individual resources (Taesombut, N. et.al., 2006). This allows a Grid to accommodate even the largest and most resource-demanding applications (Pieter, T. et.al., 2007). In the

available literature, it is noted that the most common types of grid resource include computational resources, data storage resources and the transport network interconnecting the various Grid sites. Research report (Larry, I. et.al., 2003) (Taesombut, N. et.al., 2006) on the subject hold that as the computational requirements for typical Grid application originate from the large amounts of data they need to process the transportation of this data between the involved grid resource is an important factor when it comes to cost and time efficient scheduling of the Grid's workload.

Research reports have shown that this enormous transport bandwidth requirement can best be met by optical circuit-switched transport networks. Optical circuit-switched transport networks allows for high-bandwidth end-to-end transfers capable of low latency delivery of these large amount of data, and thus are well suited to interconnect the various Grid resources. The relevance of optical networks in Grid is illustrated by the recent increase in research activities into the "super networks" (Larry, I. et.al., 2003) (Taesombut, N. et.al., 2006) (Defanti, T. et.al., 2003). Grids making use of optical circuit-switched transport networks are usually denoted as **Lambda Grids.**

A careful survey of current literature including articles, journals, and research conference reports shows that one of the areas said to be pushing the current increases in the development of Lambda grid networks is the resource intensive requirements of e-science applications. The authors (Smarr, I. et.al., 2003) (Taesombut, N. et.al., 2006) (Defanti, T.

et.al., 2003) (Banerjee, A. et.al., 2008) hold that the next generation of large-scale scientific computing applications will involve expensive resources such as super computers, storage systems, and experimental facilities, which are distributed across domains and geographical locations. Some examples of such applications, which are being developed, include the Genomes-to-life (GTL) project of the Department of Energy (DOE) USA (Elizabeth, P., .2003), Teragrid (Kate, C. 2007), and the Opt/Pater (Larry, S. 2004) project. Such projects typically require real-time transfer of gigabytes or terabytes of data from remote experiential sites and data warehouses across wide-area networks to a central computation site for data aggregation, processing, visualization, and other analysis. The argument in (Coffman, E.G, et.al., 1985) is that, this real-time transfer of huge amount of data required by such e-science applications are better addressed by Lambda grid networks.

(M. Veerarghavan et.al., 2004) worked on the Varying Bandwidth List Scheduling (VBLS) algorithm. The VBLS builds on the basic List Scheduling (LS) algorithm. In developing this scheduling algorithm the authors considered the works on admission control for blocking ahead shared resources (D. Wischik and A. Greenberg, 1998) and resource sharing for book-ahead and instantaneous request calls (A. Greenberg, R. Srikant, W.Whilt, 1999)

In the VBLS algorithm, the scheduler maintains an available capacity function $\gamma(t)$. Given it knows the Time Range Capacity (TRC) allocations for all scheduling light paths, it knows when and how much link Capacity is available for a new request. Capacity allocation for a new request is made on a round- by-round basis, where a round consists of the procedures used to allocate capacity for a time range that extends between two consecutive changes points in $\gamma(t)$. In a time range between two consecutive change points, determination is made whether the entire remaining file can be transferred or the holding time ends within this time range and whether the available capacity is greater than or less than/equal to the maximum rate requested for the number of channels.

A dedicated network is represented as a graph G= (V, E) with n node and m links, where each link $l \in E$ maintains a list of residual bandwidths specified as segmented constant functions of time. In the algorithm, a 3-triple of time bandwidth ($t_l[i]$, $t_1[i+1]$, $b_1[i]$) is used to represented the residual bandwidth $b_1[i]$ of links l at time interval $[t_l[i], t_1[i+1]]$, i=0,1, 2……., T_i-1, where T_i is the total number of time slots of link l. $t_i[0]$ denotes the current time point and $t_1[i]$ (i >1) denotes a future time point. Setting $t_i[T_1] = t \infty$ indicates that there is no bandwidth reservation on link l after $t_i[T_i-1]$ and therefore $b_i[T_i-1]$ has the full bandwidth of link l.

Taking as input graph G= (V, E) with a List of all links $l \in E$, source V_s and destination V_d file size δ, and a time slot rang [P,q], VBLS determine if there is a path from V_s to V_d such that data of size δ can be transferred within the time slot range [P, q]. where P is file transfer start time q is file transfer end time.

The following donations are used to facilitate explanation of the VBLS algorithm:

E (P, β): a subset of E consisting of links whose residual bandwidth in time slots P are less than β.

G^1 = G - E (P, β): the operation of removing the links in E (P, β) from G and producing a new graph G^{11}.

Q p: a queue storing the bandwidth of links l ϵ E sorted in a decreasing order for time slot P

The VBLS algorithm starts from time slot P and recursively calls itself by modifying the network G, advancing to the next time slot and adjusting the residual data size. Once the link with residual bandwidth less than β are removed from G, and there exists a path P from V_s to V_d in the remaining network G^1, the bandwidth of the P is at β.

For the current time P, three cases are considered: (i) if the bandwidth of Φ optical light path P is greater than or equal to the residual data size, the algorithm computes t_{end} and finishes successfully; (ii) otherwise if $P \equiv q$, the algorithm fails to find a feasible path; (iii) otherwise, the algorithm calls itself after increase data size δ. The VBLS algorithm examines all possible permutations of bandwidth at different time slots to obtain the minimum file data transfer end time.

With VBLS, before a transfer begins, the scheduler estimates the bandwidth that will be available for the entire transfer, and makes that allocation. This avoids having to

determine how much bandwidth is available during the actual file transfer (C.Jin, D.Wei, and S. Low, 2004) and (L. Brakmo, L.Peterson. 1995). The penalty (limitation) with VBLS is that network switches have to perform more bookkeeping, tracking information on file size to know when all ongoing transfer will complete, unlike in IP networks where routers do not maintain such state information. Further the added complexity of the switched is a limitation of the algorithm. First a switch needs to maintain the available capacity as a time-varying function for all interfaces, unlike in TDM/FDM switches that only maintain current available capacity information. The need for optical switches to use timer mechanisms to reconfigure the cross connection at time range boundaries for all ongoing connection increase the job blocking rate and leads to increase in file transfer scheduling delays.

(A.Banerjee, et.al 2008) worked on the Virtual Finish (ViFi) algorithm, this builds on the List Scheduling algorithm. In ViFi, before a circuit for transfer is established two scenarios may occur; either the file is completely transferred within the circuit holding time, that is referred to as initial finish time, or it is not fully transferred which is referred to as incomplete finish transfer

Scenario 1: Initial finish deals with utilization on the reserved links. When there is an early finish, the present circuit may be torn down as the lambdas that this circuit was making use of are now free and will not be in use because of the pre-allotted time slots for the entire job finish thereby causing redundancy.

Algorithm Incomplete _ File _Transfer

1) In the lines of the LFF heuristics; choose K different paths along which this file maybe transmitted.
2) The predicted transfer time is chosen as the highest transfer time (or lower transfer bandwidth) of past transfer profiles. The idea here is to avoid an incomplete transfer again.
3) Determine the path along which the file maybe scheduled at the earliest. Add this to the online schedule.

Senario11: Assume that the holding time of the current circuit may not be extended, as the links may be reserved for transfer of a different file of the same application or for a different application. For an incomplete file transfer, two different options are available; the first option is to retransmit the entire file after establishing a new circuit. The second option is to transmit only the remaining portion of the file, which could not be transmitted the first time. The former is simple to implement and also does not require any application-level fragmentation and reassembly of file components however, the time duration in which the file was being originally transmitted is completely lost. The latter requires marking of correctly transmitted sequence number. Alternatively check-pointing tools, which are available in many operating systems to maintain persistence of data and

recover from failures (Y. Wang et.al, 1995) maybe employed. Note that both approaches require establish a new circuit and hence require new link reservation to be established.

Algorithm Modify Schedule _ Early _Finish

(Circuit, Actual _ Finish _Time, Scheduling _ Finish _Time)

//All three parameters above refer to the file transferred early//

1. Consider a particular virtual link on the circuit.
2. If a file transfer is scheduled to begin at scheduled Finish _Time on this link:
    a. Identify other virtual links for this file.
    b. Check if this file maybe scheduled to start transfer between Actual _Finish _Time and scheduled _ Finish _Time on these links. If *yes*, modify the offline schedule to start file transfer at this time.
3. Repeat above steps for all virtual links in the circuit.

The algorithm does not alter the virtual link reservations which had been made by the offline schedule. It only alters the circuit start time. Moreover, if a circuit is indeed modified to start earlier than its scheduled time, the end time is kept the same. Thus circuit holding time will increase.

(Smith .W. et.al, 2000) and (Foster. I. et.al, 2000) worked on the performance of Advance reservation based scheduling as a part of Global Architecture for reservation and allocation (GARA) to guarantee resource availability at the execution time of the application. This ensures that all resource would be simultaneously available at the execution time of the application. As by reserving resource in advance one can provide an upper bound on the response time. ARs can also be used for ensuring end-to-end quality of service. For jobs with sequential tasks, the response time of the reservation for the second resource and so on, thus guaranteeing the end-to-end response time. Advance reservation have been studied in numerous contexts such as Architecture for ensuring end-to-end quality of service and for network applications. (Foster. I. et.al 2000) architecture for data-intensive collaboration, (Foster.I. et.al 2003) scheduling of data placement activities, job scheduling for clusters and supercomputers, equally Grid based architecture for dynamic optical network.

However, this work assumes that there is no laxity in the reservation window and hence rejects all incoming request that overlap with any of the previously committed reservation.

(Ranganathan et.al, 2003) studied independent CPU- allocation and data set replication through simulation, this work focuses on the allocation of data sets, but does not address network resources allocations that may be needed to access the data within a deterministic time frame.

(Bayya et.al, 2002) Market-driven and incentive-based parameter sweep applications scheduling on computational resources has been studied extensively, where resource selection policies depending on the notions of budgets and deadline are investigated. These notions influence the amount of work performed by the Grid. In contrast, this work focuses on performing all jobs submitted to the Grid (the workload demand) in a steady-state, and dimensioning the grid accordingly. Interdependent resource allocation (e.g) network bandwidth availability can influence each other and use the concept of division load to keep the combined dimensioning and scheduling problem manageable.

The effects of co-allocating CPU and network resource to a single job have been studied in (Thysebaert.P. et.al, 2004). Grid sites are connected through a VPN in which fine-grained bandwidth pipes can be setup. In reality, it is difficult to image a scenario in which such pipes with guarantees concerning delays, jitter and bandwidth availability can be setup over the internet.

In contrast, the use of optical transport network does offer the reality of high-capacity bandwidth pipes between the various grid sites and is therefore a focal point in this work.

2.3 Modeling Grid Scheduling Problem

Various literatures on file aggregation on lambda grid, one way or the other attempt at describing the problem of resource scheduling for the Lambda grid. Two things evident in these writings are the representation of lambda grid network topology as graphs and the formulation of the problem of aggregating large data files from distributed data bases

in a lambda grid network as a time-path scheduling problem (TPSP). Furthermore, available research reports (due to somewhat shot comings in TPSP) describe the modification of TPSP, calling the new problem N-destination TPSP (NDTPSP).

However, (Coffman, E.G, et.al., 1985) (Taesombut, N. et.al., 2006) (Banerjee, A. et.al., 2008] hold that both TPSP and NDTPSP (being NP-complete problems) cannot be solved in polynomial time necessitating the need for the proposition of heuristics for large scale file transmission tasks in a lambda grid.

2.3.1 Lambda Grid Network Topology Model

Graph representations of the networks topology of the lambda grid abound in the literature. A lambda-grid networks topology, an example of which is the VSN (www.net.gov/ultranet, 2007), may be represented as a graph G (V, E), where each node V represents a core switch, and the edge 'E' represents the connectivity between core switches. Core switches are connected with single or multiple lambdas (a lambda is an optical connection established over a certain wavelength). A core switch is attached to a Multi Service Provisioning Platform (MSPP).

MSPPs provide a Synchronous Optical Network (SONET)/Synchronized Digital Hierarchy (SDH) and Ethernet channels at sub-lambda granularities to end devices such as Storage Area Networks (SANs), data warehouses, or host computers. Thus, a lambda may provide an end-to-end connection between two end-to-end machines via the MSPPs

and core switches (Nageswara, S.V, et.al., 2004). The connection from the core switch to the MSPP to the end lost is not represented in graph ‘G’.

The layout of the end-to-end connectivity is shown in figure.2.1 for example; a simple way by which an end host may connect to a lambda grid is by using a Gigabit Ethernet Interface Card over a Local Area Network (LAN) connected to the MSPP. Alternatively, it may be connected via a 2.5 Gbps (OC-48) SONET connection. This connection from the MSPP to the end host is termed as a sub-lambda connection. It is suggested in (Banerjee, A. et.al., 2008) that in order to simplify the problem setting, that the assumption should be made that all end hosts are connected to the MSPPs with the same connection bandwidth (that is 1, or 2.5 Gbps), and therefore, the granularity of each sub-lambda connection is the same.

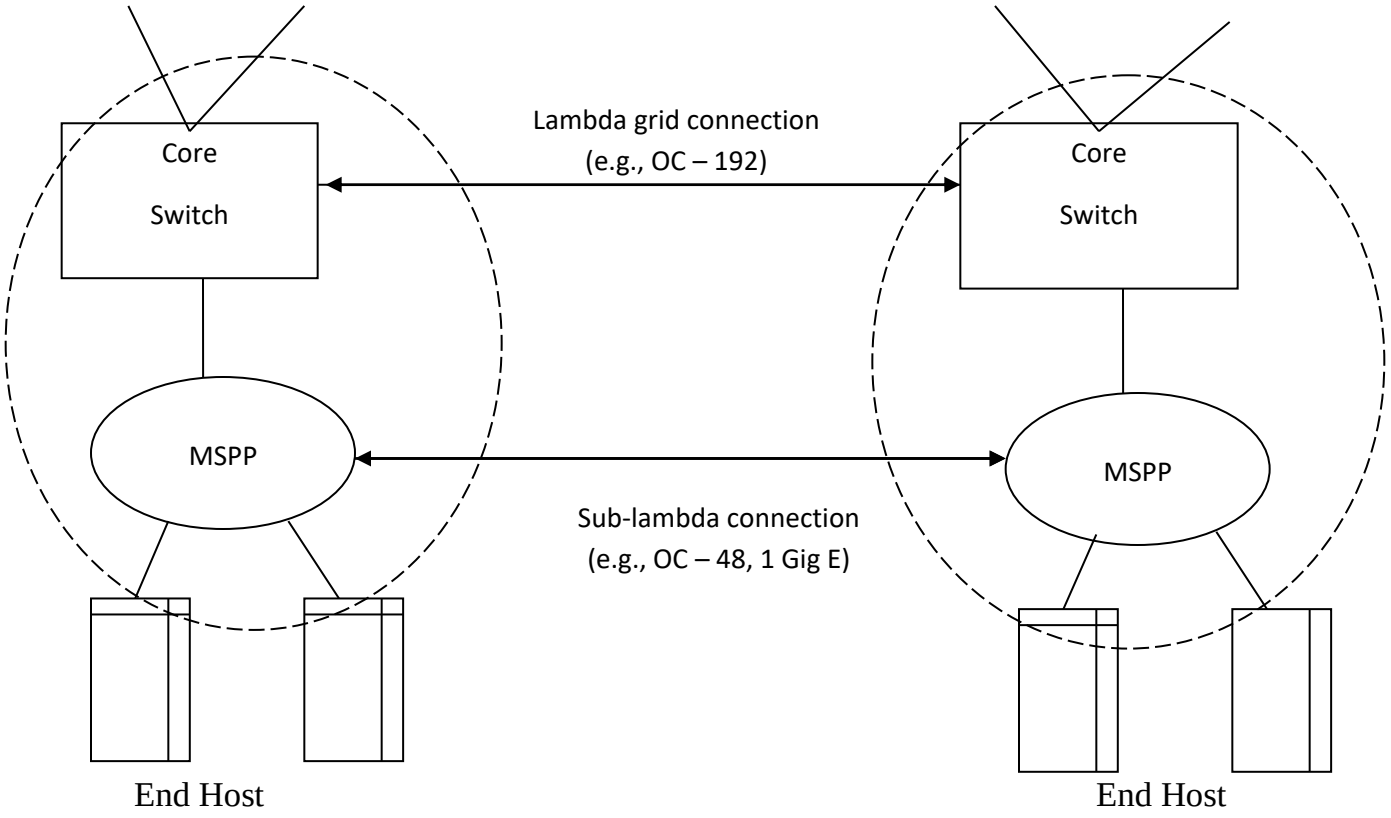

Figure: 2.1 Illustration of a Lambda Connection between two end hosts deploying lambda grid

2.3.2 Data Aggregation Problem Formulation in a Computational Grid Network

Discussed extensively in the literature is the mathematical representation of the problem of data aggregation in a lambda grid. In (Shen, S. et.al., 2008) (Pieter, T. et.al., 2007) the problem of aggregating large data files from distributed databases in a lambda grid networks had been formulated as a Time-path Scheduling Problem (TPSP).

An illustration of the problem formulation on a six-node network is shown in figure 2.2 below:

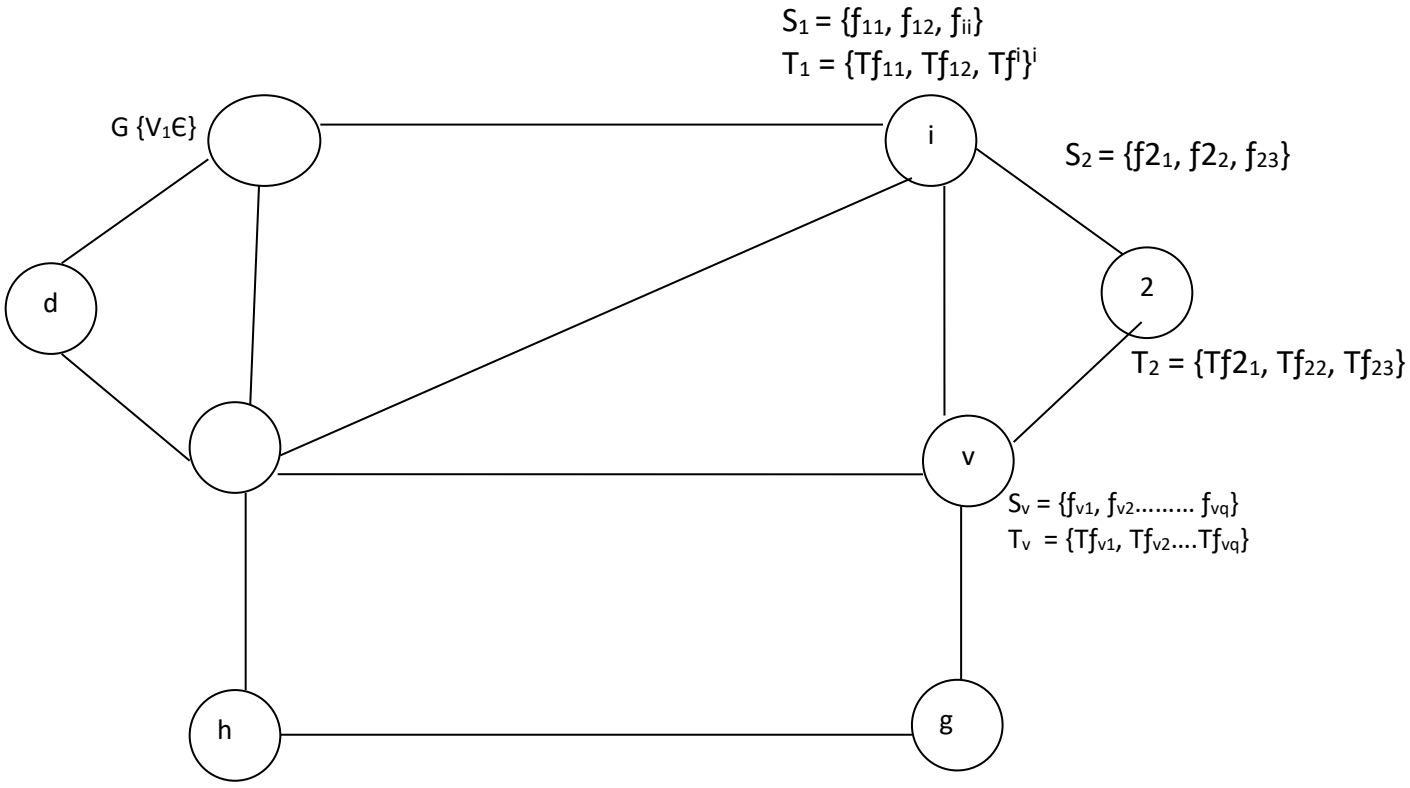

Figure 2.2 Problem formulation of TPSP on an example eight-node network.

Though differences exist mainly in notation used by different authors, the mathematical representation is given as follows;

At each core switch v∈V, there exists a set of files $S_v = \{f_{v1}, f_{v2}.....f_{v2}\}$ corresponding to the end hosts that it is connected to, whose estimated transfer time over the lambda grid to the destination, d, T_f is known and is denoted by the set $T_v = \{Tf_{v1}, Tf_{v2}.....Tf_{vl}\}$, were Tf_{vl} is the transfer time for file fv_1. The objective of this problem formulation as understood in the literature is to determine the following:

1. **Route:** This is the path on the lambda grid, via, which a file should be transferred from the source to the destination.
2. **Time Schedule:** This is the time at which a connection must be reserved on the lambda grid for the corresponding file. This is important because it may not be possible to transfer all the files simultaneously on the lambda grid due to link capacity constraints.
3. **Minimum finish time:** The objective is to minimize the total time required to aggregate all the data by using the lambda grid. The last file to reach the destination may be the bottle neck for the super computer to connect to the MSPP (see figure. 2.1). Since computation cannot be completed unless all the data is aggregated.

In a lambda grid, a super computing machine has high bandwidth connection to the MSPP (see Figure. 2.1) and, thus, has access to all connections arriving at the MSPP that is connected to. The nodes on the graph to which the super computer is attached is marked as d€v. At a certain step in the computation, the super computers may require data aggregated from multiple end hosts (data warehouses, SANs etc) before it resumes computation. This process is modeled as the transfer of files from each and host to be destination super computer. All the data that must be transmitted from one end host is modeled as one file.

The literature showed that the two dimensions of determining both the path and time schedule make TPSP as a problem exceptionally hard, and it differentiates TPSP from other machine scheduling problems that have been reported in the literature (Pinedo, M.L., 2008). However, proves reported in the literature show that TPSP is NP complete by reducing it to the Multiprocessor Scheduling Problem (MSP) (Garey, M.R and Johnson, D.S, 1990).

2.4 Heuristics for Routing and Scheduling

As recorded in the literature, TPSP is an NP-complete problem which cannot be solved in polynomial time (Banerjee A. et.al., 2008) (Garey M.R and Johnson, D.S, 1990). And in consequence heuristics should be proposed for large scale file transmission tasks.

Extensively discussed in the literature is the proposition of a greedy approach for solving TPSP. The greedy approach chooses one file at a time and determines the route along which this file may be scheduled at the earliest. The file is scheduled along this route. Though a number of heuristics is documented in the literature for choosing the best file, two heuristics seem to stand out. These are the Largest File First (LFF) and the Most Distant File First (MDFF). Similarly, a number of algorithms are recorded in the literature for determining the route and schedule. However, most papers extensively described. ‘All Possible Time Slots (APT) Algorithm and K-Randomized Paths (KRP)’.

2.4.1 Largest File First (LFF)

This approach is based on the intuition that the largest file (having the largest estimated transfer time) is the bottleneck for scheduling because it requires more resources in terms of the amount of time required to be free on the links of the lambda grid. Thus, the largest file remaining to be scheduled is picked as the greedy choices.

2.4.2 Most Distant File First (MDFF)

This approach is based on the intuition that files are located at nodes far away from the destination in terms of number of hops must be given higher priority for scheduling because they require more links to be free for files to be transferred. Files are chosen in the order of the number of hops that they are located away from the destination.

2.4.3 Algorithms to Determine Route and Schedule

After a file ‘F’ is chosen using one of the above heuristics, it may be routed and scheduled on the lambda grid by using one of the following algorithms.

2.4.4 All Possible Time Slots (APT) Algorithms

This Algorithm first computes all time slots that are available between the file source [N(f)] (denoted as source S) to the destination 'd' for the duration estimated for transferring file f (Tf).

The technical Community recommends the use of the bandwidth scheduling algorithm reported in (Nageswara S. V, et.al, 2004) which is based on the Bellman-ford shortest path algorithm (Thomas H. C., et.al., 2014) applied to the disjoint time intervals at which the links are available. The algorithm is described in brief in Figure 2.3.

Algorithm: find – All-possible – Time-slots (source, destination, duration, v)

1. Initialize
   L [i, j] = { Set of available time intervals of long greater duration of i and j are connected. Null set Ø after wise.

2. $D(s) = \{R^T\}$, $D(V) = \{\emptyset\}$ $V \quad v \neq s$

3. For K = 1, 2……., |V| do

   for each edge (u, v) do

   $D(v) = D(v) \oplus (D(u) \otimes L(U,V)]$ —

4. return D(d)

Figure .2.3: Bellman-ford Algorithm to determine all possible time slots.

Where $\oplus$ denotes margin, and $\otimes$ denotes inter section 2 lists.

If time slots of duration T_f or greater are available before the current finish time, then best fit available time slots is chosen, or else, the earliest available time slot is chosen.

Site *f* is scheduled on the chosen time slots and routed along the corresponding path.

The complexity of the algorithm described in figure 2.3 may be written as;

$O(|V| * |E| * (O)\oplus) + O(\otimes)]$. where $O(\oplus)$ and $O(\otimes)$ are function of the numbers of disjoint time intervals on the links ($\oplus$) denotes the operation of merging the disjoint time interval and ($\otimes$) denotes the operation of interaction of the disjoint time intervals).

2.4.5 K. Randomized Paths (KRP) Algorithm

This algorithm chooses the best path among K randomly chosen paths. The steps are outlined in figure 2.4. It is important to choose random paths because if a fixed set of paths are chosen (for example, K shortest paths), then a few links in the lambda grid may get increasingly congested and the finish time may be poor.

Algorithm: K-Randomized Paths (source, destination, *f*,*v*,k)

1. Find random K-alternate paths from the source to destination. Random K-attracts paths may be achieved by randomly picking weights of the links and applying Djikstran's Algorithm to compile shortest path (Nageswara S. V, et.al 2004).

2. Out of these K paths, choose the one in which file f may be scheduled at the earliest.

Figure.2.4 K-Randomized Path Algorithm.

.

The complexity of step 1 may be stated as O (K*|v| log |v|), since for a suppose graph, (|E| < |v| log |v|) and the complexity of Djikistra's algorithm is O (|v| log |v|). The complexity of step 2 may be written as O (k*|v|), since |v| is the maximum length of a path in a graph, and we assume that the cost of merging the disjoint time intervals is a constant. The overall complexity of the algorithm is O (K *|v| log |v|). Typically, the number of alternate paths that needs to be chosen is much less than the number of vertices (K << |v|). Therefore, the complexity is O (|v| log |v|),

2.5 Performance Evaluation Metrics for Lambda Grid Scheduling Algorithm.

The references (Veeraraghavan, M. et.al., 2003) (Michael L. Pinedo, 2008) (Garey M.R, and Johnson, D.S, 1990) (Thomas H. Cormen, et.al., 2014) indicate that the performance of lambda grid resource scheduling algorithm are evaluated based on metrics such as job blocking rate, fairness and effectiveness;

- Job blocking rate – it is the percentage of jobs blocked divided by the total number of jobs submitted.
- Fairness – it is the metric that shows performance of the heuristics for smaller and large jobs.

- Effectiveness – it is calculated as the percentage of latest finish time of the job scheduled and the blocking rate of the maximum time slots S. The higher the percentage, the more effective the heuristics.

Effectiveness = [1 – (Job Blocking Rate X Latest Finish Time of the Jobs Submitted)/ (100% x maximum Timeslot)] x 100…… (2.1)

2.6 Summary of Related Literature and Research Gaps

(Liu, X. et.al., 2009), proposed an algorithm that uses Deadline constant for task scheduling and light path establishment in the lambda grid. This was reported to be effective in minimizing the lambda grid resource usage and improving file aggregation time. However, in (Page, A.J, et.al., 2005),

- It was considered unviable since it is based on the assumption that edge node and computing node have unlimited buffer to store sets which cannot be realized in practice. ***__This will be solved using the exchange of information on the transmission task Analyzer and Link Resource Analyzer of the proposed.__

Work done by the author (Castillo, C. et.al., 2007), focused on the use of advanced reservation of resources using Best Fit strategy for lambda grid resource scheduling. Results obtained indicate a substantial improvement of grid resource utilization. However as reported in reference (Lakshmiraman, V. and Ramamurity, B. 2009), the job blocking

rate based on this scheduling technique increases (though marginally) with every newer scheduling cycle.

- The shortcoming of this technique is that the job blocking rate based on this scheduling technique increases with every newer scheduling cycle. *****This will be resolved using the network state information module of the proposed.**

In (Ho, P.H and Mouftah, H.T, 2003), the authors proposed a grid scheduling technique which is based on exchanging information about critical optical paths (i.e. links) in the network and avoiding those links during wavelength assignment. It was shown that this method reduces the blocking probability compared to a fixed wavelength assignment scheme. However, the limitation of the scheduling method has to do with the overhead imposed by the exchange of network link-state information.

- Limitation of this scheduling method has to do with the overhead imposed by non-exchange of network link state information. ****Resolved by the Adaptive scheduler module of the proposed.**

(Coffman E.G et al 1985) Considers scheduling file transfers over network when file size and the maximum numbers of file transfers possible from each node are given.

- Considers a fully connected mesh network, but lambdas have sparse connectivity. *****This will be resolved by automatic re-provisioning of idle lambdas.**

(Nageswara, S.V. et.al., 2004), Proposed scheduling algorithm that compute the quickest path with a minimum end-to-end delay to transfer file of a given size from its source to destination when bandwidth and delay constraints on the links are specified. Using similar assumption, a Varying Bandwidth List Scheduling (VBLS) heuristic to compute circuit over a lambda grid was studied in (Veeraraghavan, M. et.al., 2003). (Buyya, R. (ed), 1999) (Pieter, T. et.al., 2008) showed that the basic problem with most of these grid scheduling algorithms is that they assume an ideal communication system where all the resource is fully connected and communication between two resource can be used whenever needed.

(Garey, M.R et al 1979), (Hall, N G et al 1996), Machine scheduling modeled as single processor scheduling, multi-processor, open-shop, flow-shop, and job-shop scheduling.

- The problem, however, is that it is not applicable to the proposed problem since the resource Lambda are not independent, rather, they have connectivity relationship among them. ****__Resolved at the layer 4 framework of the proposed.__**

(M. Veerarghavan et.al., 2004) The penalty (limitation) with VBLS is that network switches have to perform more bookkeeping, tracking information on file size to know when all ongoing transfer will complete, unlike in IP networks where routers do not maintain such state information. Further the added complexity of the switched is a

limitation of the algorithm. First a switch needs to maintain the available capacity as a time-varying function for all interfaces, unlike in TDM/FDM switches that only maintain current available capacity information. The need for optical switches to use timer mechanisms to reconfigure the cross connection at time range boundaries for all ongoing connection increase the job blocking rate and leads to increase in file transfer scheduling delays.

- The problem is that they assume an ideal communication system where all the resource is fully connected and communication between two resource can be used when ever needed. ******Resolved using the Adaptive scheduler module of the proposed Algorithm****

(A.Banerjee, et.al 2008) pitfall of ViFi is that the algorithm does not alter the virtual link reservations which had been made by the offline schedule. It only alters the circuit start time. Moreover, if a circuit is indeed modified to start earlier than its scheduled time, the end time is kept the same. Thus circuit holding time will increase.

CHAPTER THREE

MATERIALS AND METHOD

3.1 Chapter Overview.

This chapter presents the method that would be used in achieving the work, the reason for the particular method to be used and design of the adaptive lambda grid resource scheduling algorithm. The lambda grid scheduling problem is formulated as a time-path scheduling problem (TPSP). The scheduler is composed of three main algorithms. Algorithm 1 is run to allocate lambdas; the function of algorithm 2 is to determine the file transmission path in the lambda grid. Algorithm 3 implements the lambda grid resource scheduler. The scheduler is the module that schedules the actual file transfer. Every run of the path determination algorithm (Algorithm 2) is integrated with the running of the lambda allocation algorithm i.e (Algorithm 1).

The scheduler iteratively runs the path algorithm to dynamically re-establish the shortest path from the source to the destination (with the consequent re-allocation of wavelengths). This reallocation of wavelengths ensures that idle lambdas can be re-provisioned for ongoing or later file transfer task in the lambda grid.

3.2 Description of Materials/Tools Used

- ✓ Cisco Packet tracer 7.1
- ✓C++ Source Code
- ✓CORBA Protocol
- ✓System Model: Hp Elite book 8440P, Intel® core™ i7 cpu, Memory 6144MB RAM, Page file 5862.
- ✓Matlab m-file.
- ✓Operating System: Windows 7 Ultimate 64 bits (6.1, Build 7600)

3.3 Methodology

Methodology is the set of technique/procedures that is followed in the analysis and or the design of a system. In the design of the proposed lambda grid scheduling system graph theory is used in the representation of the lambda grid network topology, modeling and simulation was used for the result presentation. The proposed scheduling algorithm is formulated as a time path scheduling problem. The schedule adaptation is realized in the algorithm by varying the allocation of lambdas. This technique allows the proposed scheduler to backfill any holes left in wavelength allocation from other request. This inefficient is a reallocation of idle lambdas i.e. effectively adjusting the schedule.

The design will be implemented into computer program using the C++ programming language with MATLAB m-file code integration for the simulation of the lambda grid network to test the performance of the proposed lambda grid scheduling system, the 24-node National Lambda Rail would be used as case study lambda grid network. Details of this will be presented in the simulation to be carried out in the next chapter from the results of the simulation; the performance of the proposed algorithm will be compared with those of the Varying Bandwidth List Scheduling algorithm and the Virtual Finish First algorithm. The adopted methodology is dependent on the nature and scope of the system project.

3.3.1 Graph Theoretic Methods.

This method is also simply called graph theory or network analysis. The methodology employs the use of graphs. Graphs are mathematical objects that can be used to model networks, data structure, process scheduling, competitions and a variety of other systems where the relationship between the objects in the system play dominant role. (Olaf Sporns, 2010)

3.3.2 Method Used.

Graph theory is adopted for the systems analysis, then modeling and simulation for result presentation. The system fundamentally being a network scheduling and routing algorithm makes graph theoretic method most suitable for its analysis. Graph theory is best suited to the analysis and development of network routing algorithm (Bart, S. et.al., 1992). Furthermore, specifically the formulation of the TPSP problem is basically based on graph algorithm. (Banerjee et.al., 2004)

3.4 Analysis of Existing System.

The depiction and analysis of the existing grid scheduling algorithm is facilitated by the use of the graph and the pseudo code for the List scheduling algorithm.

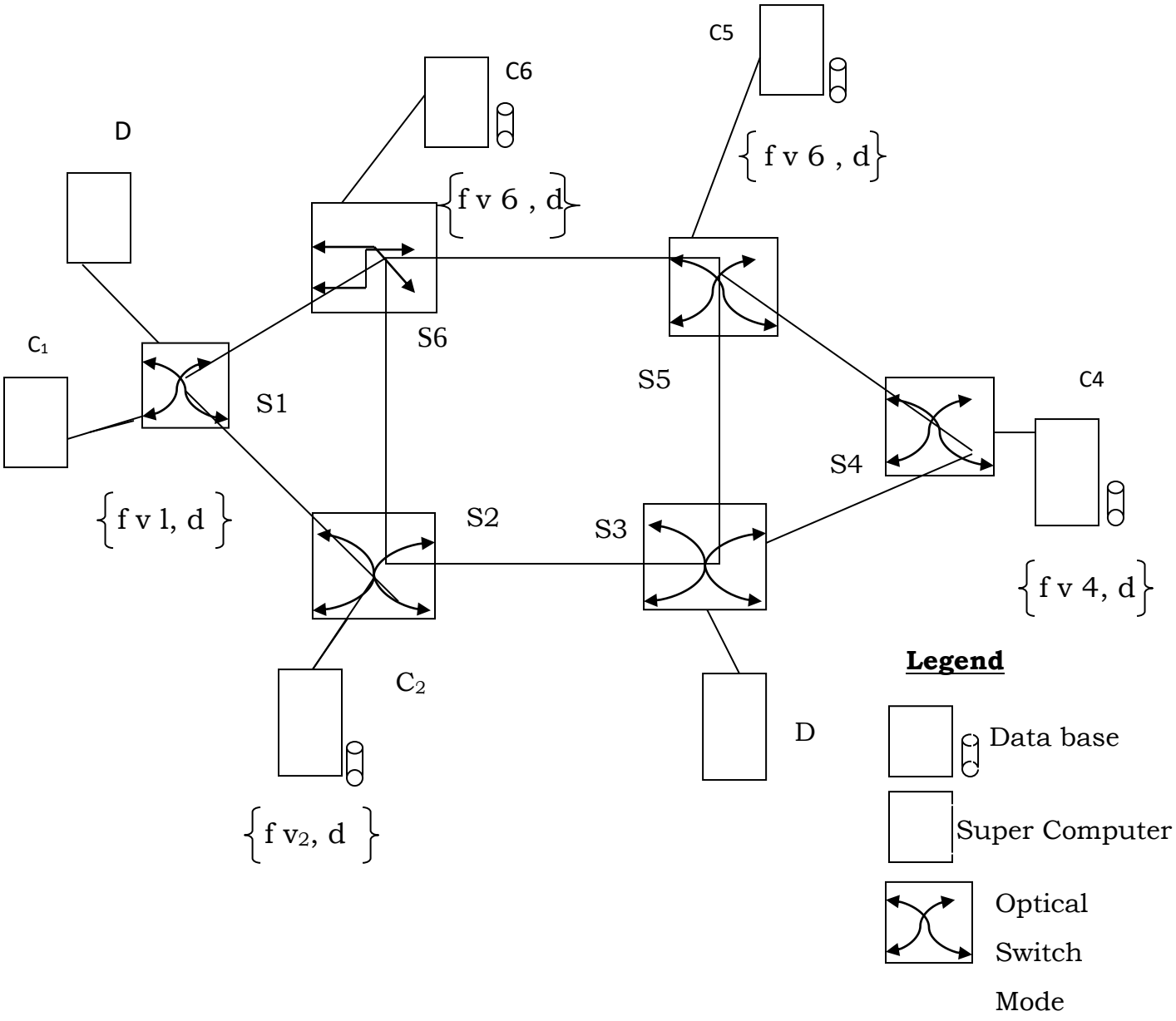

Figure 3.1 Analysis of Existing Lambda Gird Scheduling Algorithm. (Pinedo M.L.2008)

The optical network of lambda grid with attached computational and storage resources is denoted by the graph G (V, L) as shown in the figure.3.1, V and L represent nodes and optical links in the graph respectively. V can be denoted as {S, C, D} in which S, C and D are respectively sets of switch nodes, computational resources and databases, each link $\iota \epsilon$ L support only one single wavelength capacity C. Databases and super computers are directly connected to switch nodes. At databases D there exist a set of files to be aggregated to different computational resources C. Access networks connecting databases and computational resources to switch nodes provide large enough bandwidth which can be shared simultaneously by many transmissions. Thus switch directly connect with databases and super computers can be seen as the source and destination for every file transmission task.

The existing grid scheduling algorithm is the LIST scheduling algorithm. As shown in the figure 3.2, the existing LIST scheduling algorithm includes two main steps: list and schedule. In step 1LIST, all file transmission tasks are sorted into a scheduling list LIST according to certain priority scheme. Step II schedule is loop of three sub-steps (1), (2), (3) shown in figure 3.2. Route and time scheduling of a certain file transmission task can be determined independently.

I. LIST transmission tasks in LIST

11. Allocate resource:

do

(1) Select a task from LIST
(2) determine route
(3) determine time schedule

loop while there is unscheduled task**.**

Figure. 3.2 Process of the Existing LIST Scheduling Algorithm. (Micheli, De. 1994).

The algorithm iterates over the LIST. An unscheduled task is selected from the LIST with the highest priority of Longer file first. And then allocate optical link resources which build up its route for time interval equal to f^n, s transmission time, as to determine the time interval for f^n. strategies for steps and sub-steps:

STEP I: List priority scheme for longer files scheduled earlier – longer file first (LFF). Sort all transmission tasks by their aggregation request arriving time – First Arrive First Serve (FAFS).

STEP II: This sub-step just selects transmission tasks with the highest priority from LIST, it is implemented independently. Along with independent Step II (1) the loop

operation makes the algorithm iterates over the LIST and do a sequential scheduling for the list.

STEP II (2): The routing scheme is determined i.e. routing schemes generally used for light path establishment in optical networks: fixed Routing (FR), Fixed Alternative Routing (FAR).

STEP III (3): Most commonly used scheme for time scheduling of transmission tasks is Idle Time Interval Insertion (ITII). For a certain transmission task selected from LIST the algorithm searches potential idle time internal equal to or longer than task transmission time.

3.4.1 Limitation of the Existing Algorithm

Does not use network state information at the time of light path establishment, hence step II (2) of the list algorithm cannot be implemented as adaptive routing scheme.

- Does not minimize delays in data aggregation due to the high job blocking rates of the algorithm.
- Does not dynamically readjust the schedule online during the actual file transfer to accommodate the actual amount of time that is required to transfer a file.
- Does not automatically provision idle lambdas to service current resource request from grid applications.

- Does not provide more time within the schedule for incompletes file transfer event-since the online reconfiguration module of the algorithm does not attempt to adjust the schedule of the next file transfer – making the algorithm somewhat behave like the best effort format of step111(3).

3.5 Data Analysis.

The analysis of the lambda grid data is done to find parameters for accommodating the variance in file transfer times. (in order to predict optimal circuit holding time). To do this different number of standard deviation (σ) away the mean (m) (i.e. mean of past data transfers), which would correspond to the upper limit of a confidence interval in a normal distribution is computed.

An analogy of the situation, calculating two points and bearing configuration of Nnamdi Azikiwe University Awka and Nnewi Teaching hospital was done using the link of Nigeria Communication Commission NCC project for federal Universities and their Medical Teaching Hospital.

Distance Calculation between two points on earth.

This uses the '**haversine**' formula to calculate the great-circle distance between two points – that is, the shortest distance over the earth's surface – giving an 'as-the-crow-flies' distance between the points (ignoring any hills they fly over, of course!).

Haversine formula:

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

Where

φ *is latitude,* λ *is longitude,* R *is earth's radius (mean radius = 6,371km);*

note that angles need to be in radians to pass to trig functions!

The haversine formula[1] 'remains particularly well-conditioned for numerical computation even at small distances' – unlike calculations based on the *spherical law of cosines*. The '(re)versed sine' is $1-\cos\theta$, and the 'half-versed-sine' is $(1-\cos\theta)/2$ or $\sin^2(\theta/2)$ as used above. Once widely used by navigators, it was described by Roger Sinnott in *Sky & Telescope* magazine in 1984 ("Virtues of the Haversine"): Sinnott explained that the angular separation between Mizar and Alcor in Ursa Major – 0°11′49.69″ – could be accurately calculated on a TRS-80 using the haversine.

For the curious, *c* is the angular distance in radians, and *a* is the square of half the chord length between the points.

If `atan2` is not available, *c* could be calculated from $2 \cdot \text{asin}(\min(1, \sqrt{a}))$ (including protection against rounding errors).

Using Chrome on a middling Core i5 PC, a distance calculation takes around 2 – 5 micro-seconds (hence around 200,000 – 500,000 per second). Little to no benefit is obtained by factoring out common terms; probably the JIT compiler optimises them out.

NOTE:

This is adopted from https://www.movable-type.co.uk/scripts/latlong.html

You can find other calculations on that site and the java script for doing this calculation is also freely available on that site.

USED Bearings

Nnamdi Azikiwe University, Awka; Latitude**: 6.247826º East**

Longitude: 7.117259º North

Nnamdi Azikiwe University Teaching Hospital, Nnewi; **Latitude: 6.026075º East**

Longitude: 7.913929º North

Calculated distance between Nnamdi Azikiwe University, Awka and Nnamdi Azikiwe University Teaching Hospital, Nnewi = **33.37KM.**

Direction of Nnamdi Azikiwe University Teaching Hospital, Nnewi from Nnamdi Azikiwe University = **222.37 degrees.**

Direction of Nnamdi Azikiwe University from Nnamdi Azikiwe University Teaching Hospital, Nnewi = **42.34 degrees.**

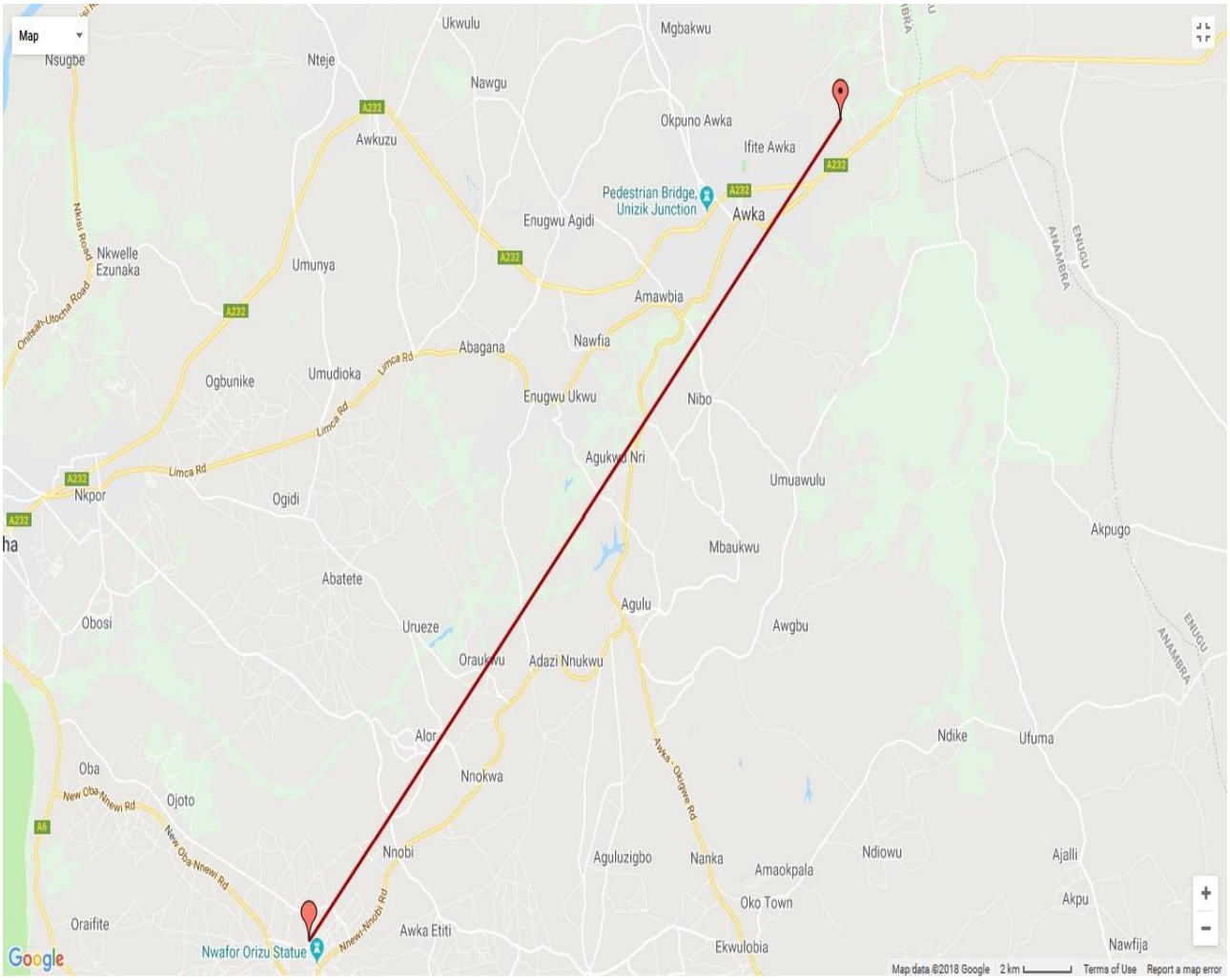

Figure 3.3: Straight Line between two Points (Unizik and Nnewi Teaching Hospital)

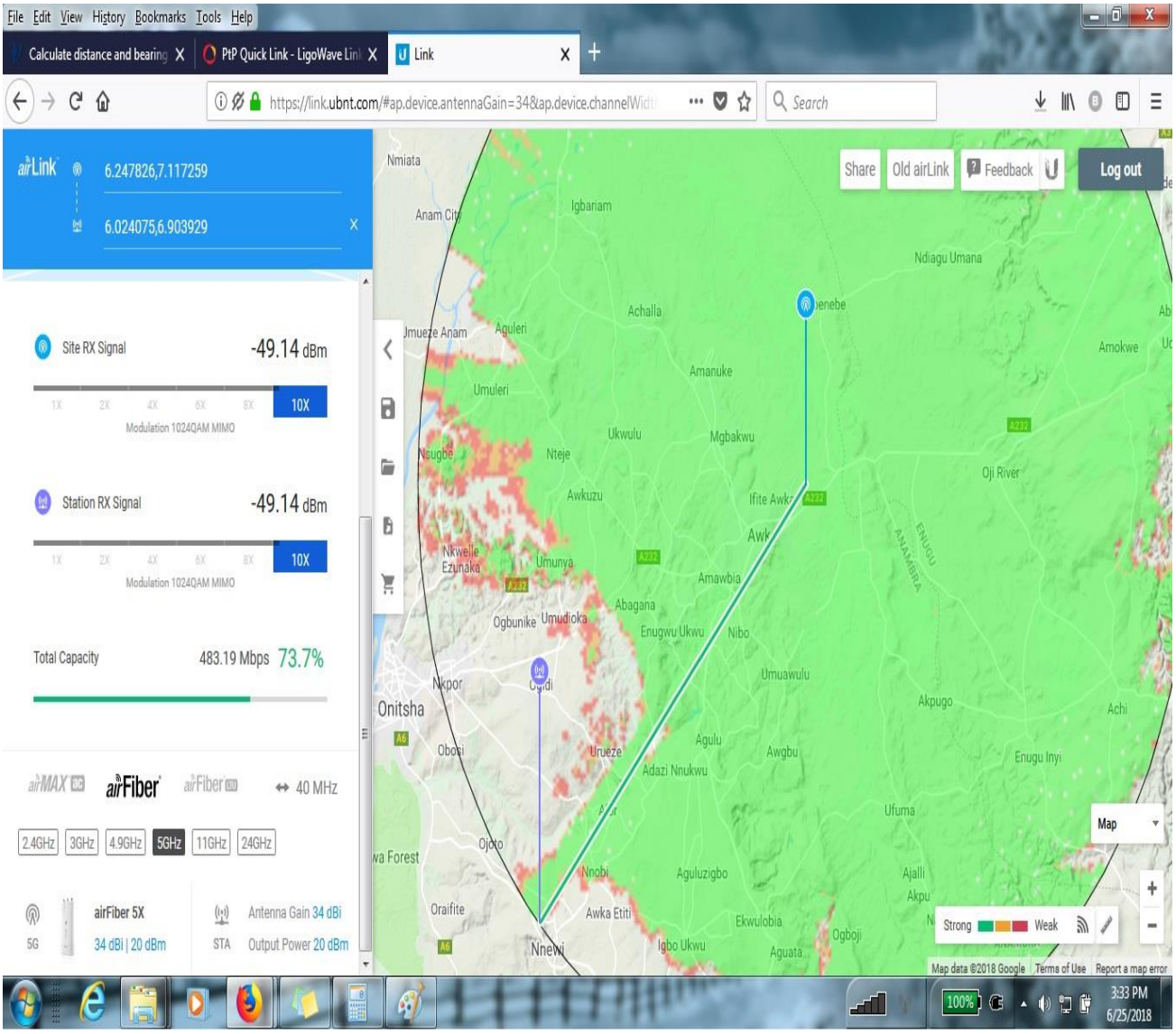

Figure 3.4 Bearing coordinate of Nnewi Teaching Hospital and Unizik via Lambda link

Analysis included estimating the following parameters.

- **Targ:** average offline schedule finish time
- **Tmax:** Maximum observed actual finish time in N transfers

- **Nmax:** Number of incomplete files transfers when Tmax was measured.
- **Tmin**: Minimum observed actual finish time in N transfers.
  - **Nmin**: Number of incomplete files transfer when Tmin was measured.
  - **Job Blocking Rate:** Percentage of jobs blocked divided by the total number of jobs submitted.
  - **Fairness:** Metric that shows performance of the heuristic for smaller and larger jobs.
  - **Effectiveness:** It is calculated as the percentage of latest finish time of the job scheduled and the blocking rate to the maximum time slot S.

3.6 DATA / INFORMATION GATHERING

The Lambda grid being the next generation of scientific computing platform is mainly found at government science project sites around the world – the developed economies. Technical data on the operations and problems of the lambda grid is best obtained from report and review documentation. Hence the data gathering technique for this work entails review of documentation on the HS department of Energy's Ultra-Science Net (DOE: Ultra Science Net Test bed. www.csm.ornl.gov/ultranet/overview.pdf), the National Lambda Rail (NLR) networks (Doug, H., 2003), and the Teragrid now replaced with Extreme Science of Engineering Digital Environment (XSEDE) (Travostino, F., et.al., 2006). The data obtained from the lambda grid project document are:

- The file transfer profiles at each transmitting node to super computers.
- File sizes and associated transfer times in the gird.
- Mean transfer time
-
- Link utilization and link capacity (OC – 192 ⟶ 10Gbps, each sub lambda OC – 48 ⟶ 2.5Gbps
- Sample lambdas grid network topologies.

3.6.1 Design of the Proposed System.

The proposed grid resource scheduling algorithm is conceptualized to overcome the pitfalls in the existing LIST scheduling algorithm for the lambda grid network.

The proposed algorithm is to meet these specifications:

- Will dynamically readjust the scheduler online during the actual file transfer to accommodate the actual amount of time that is required to transfer a file.
- Speeds up data aggregation by minimizing the blocking probability of the schedule window with the lambda grid network.
- Automatically re-provision idle lambdas to service current request from lambda grid nodes.
- Uses network state information at the time of light-path establishment: uses adaptive routing in the implementation of step 11 of existing LIST algorithm. In this algorithm (1),(2),(3) of step11 in the existing LIST algorithm will be implemented.
- Uses idle time interval insertion (ITII) scheme for time scheduling of transmission tasks. This ensures the provision of more time within the scheduler for incomplete file transfer event i.e enabling the file aggregation process to recover from file transfer error by re-adjusting the scheduler of the next file transfer.
- Supports advance reservation request. i.e upon request from grid nodes come scheduler lambdas and sub-lambdas in advance.

BLOCK DIAGRAM OF THE PROPOSED LAMBDA GRID SCHEDULING ALGORITHM.

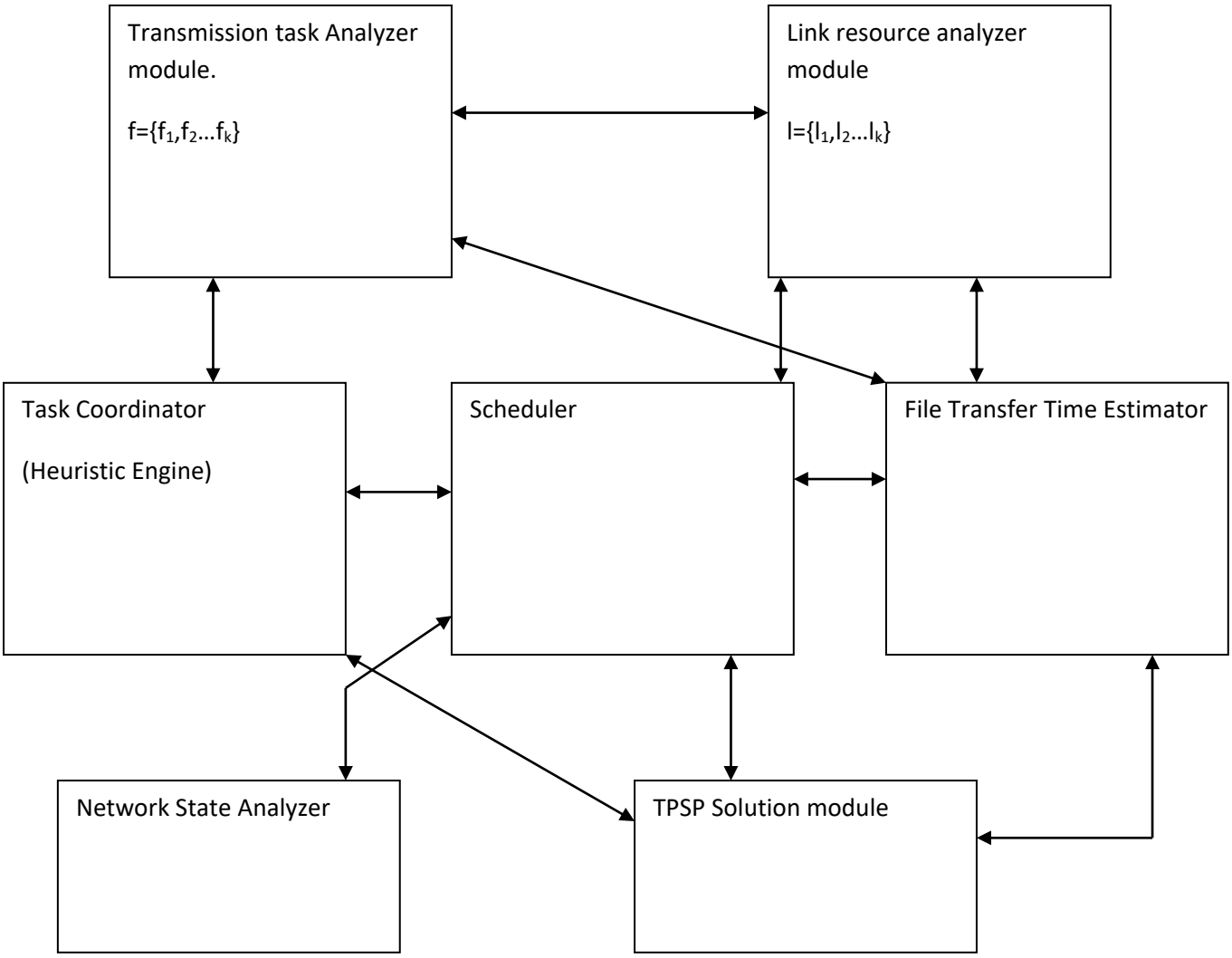

Figure.3.5: Block Diagram Overview of the Proposed Adaptive Lambda Grid Scheduling Algorithm.

Referring to the block diagram, the transmission task analyzer module carries out computation to find all transmission task in the lambda grid, sort all transmission tasks by their aggregation request arriving time and builds a transmission task LIST i.e $\boldsymbol{f}=\{ f_1, f_2, f_3\text{-------} f_k\}$. where f= set of file to be transferred in the grid.

The link resource analyzer module computes for available lambda resource to be scheduled. It analyzes the grid structure, finds the available link and builds a link set. The transmission analyzer handshakes with the link resource analyzer module to find resource requirements for file transfer task.

The task coordinator provides an abstraction of the task; it exchanges data with the task analyzer module using parameter passing. It evaluates and set file transmission task priority based on heuristics. Due to the difficulty of the TPSP problem solution, this engine loads and runs the heuristics function steps- based on the nature of the task.

The network state analyzer analyses the state of the network for latency, delays, hops, events, exceptions and dynamically builds the link state information table.

The scheduler rates over the LIST built by the combination of the transmission task analyzer and task coordinator object (i.e the heuristic engine) communicates with the link resource analyzer (i.e the routing algorithm), the TPSP solution module, the network state analyzer (in order to decide route based on network state information at the time of light-path establishment) and uses the offline schedule (i.e the outcome of the TPSP solution module) to schedule the file transfer on an reserved lambda grid link corresponding to the schedule. Adaptive technique is proposed for the schedule to dynamically adapt the offline schedule to accommodate the actual file transfer time.

3.6.2 Modeling the proposed Lambda Grid Scheduling Problem

A lambda grid network topology can be represented as a graph *G(V, E)*, as shown in Figure 3.6. Vertices *V* represent switch nodes, and the edges *E* represent optical links connecting the switch nodes. The assumption is made here that all the optical link has the same capacity C (say OC-192), and support a single wavelength. Each data warehouse (database) and supercomputers are connected to switch nodes through dedicated link of capacity C

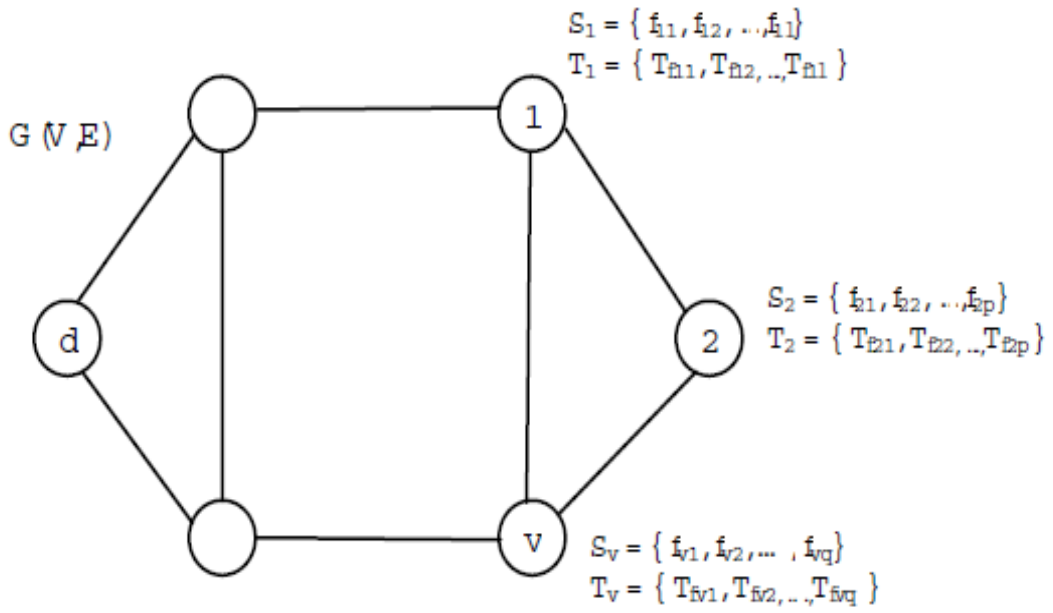

Figure 3.6 A Lambda Grid Network Topology. (Banerjee A, 2004)

At a certain step in the computation, the supercomputer may require data aggregated from multiple data warehouses before it resumes computation. This process is modeled in this design as the transfer of *files* which require to be sent from the source switch node (to

which the corresponding data warehouses are connected) to the destination supercomputer. A query is first issued by the supercomputer to all data warehouses to determine the file size required from each warehouse. The file size provides information on its expected transmission delay. The time that it takes to transfer a file along a route is:

$$T_f = S_f/C + P_d + O_c \quad \text{.....(3.1)}$$

Where T_f and S_f denote the total file transfer time and the size for file ***f***, respectively, and ***C*** denotes the capacity of each link.P_d is the maximum value of end-to-end propagation delay, and O_c is the control packet overhead. At each switch node ***v,*** there exist a set of files S_v=$\{f_{v1}, f_{v2}, \ldots\ldots f_{vl}\}$ whose T_f is pre-computed and denoted by the set T_v= $\{T_{f_{v1}}, T_{f_{v2}}, , , , , T_{f_{vl}}\}$ (shown in Figure 3.4). The node the supercomputer is connected is as ***d***, where all the files are destined to.

The objective is to determine the following:

1) ***Route***: The path through which a file should be transferred from the source to the destination.

2) ***Time schedule***: The time at which a file has to be transmitted in a single burst so that it can be transferred through the route determined in Step 1. This is important because two files which share a link on their routes should not be transmitted at the same time to avoid collision due to the constraints defined for the scheduler.

The aim is to minimize the total time for data aggregation. This is assuming that the last file to reach the destination is indeed the bottleneck, since computation cannot begin unless all the data is accumulated.

The objective of the system design is to determine Route and time schedule given the following:

1) Set *R* of switch nodes in the network.

2) The destination switch node at which the supercomputer is located, *d*.

3) Set *M* of files which have to be transferred to the destination *d*.

4) Physical-connectivity adjacency matrix, $P(i,j),\ \forall i,j \in R.\ P(i,j)$ takes two values, 0 and 1. $P(i, j) = 1$ denotes connectivity.

5) Switch nodes at which the files are located (and to which the corresponding data warehouses are connected), $N(m) \in R,\ \forall m \in M$.

6) Transfer Time (T_f) for each file, $T(m) \in Z+,\ \forall m \in M$. This can be pre-computed using Equation (3.1).

Subject Variables:

1) Virtual-connectivity matrix, $V\ mi,j\ ,\ \forall i, j \epsilon R, m \epsilon M$, takes two values, 0 and 1. $V\ m\ i,j = 1$ denotes that the file *m* is routed along a path which contains the link from *i* to *j*.

2) Start time $\tau(m)$, $\forall m \in M$, denotes the time at which the file m is transmitted. File m is transferred along the determined route from time $\tau(m)$ till time $\tau(m) + T(m)$.

Constraints:

1) *Connectivity constraints*: These constraints ensure proper virtual connectivity.

$$V_{i,j}^{m} \leq P_{i,j} \quad \forall i,j \in R, m \in M \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.2)$$

$$\sum_{j=1}^{j=|R|} V_{N(m),j}^{m} = 1 \quad \forall m \in M \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.3)$$

$$\sum_{m=1}^{m=|M|} \quad \sum_{j=1}^{j=|R|} V_{jd}^{m} = |M| \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.4)$$

$$\sum_{j=1}^{j=|R|} V_{x,j}^{m} = 1 \quad \forall x \in R, \forall m \in M \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.5)$$

$$\sum_{j=1}^{j=|R|} V_{j,x}^{m} = \sum_{k=1}^{k=|R|} V_{x,k}^{m} \quad \forall x \in R, m \in M'' \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.6)$$

Constraint (3.2) ensures that a virtual link may exist only if a physical link exists. Constraint (3.3) ensures that a virtual link must exist from the source node of each file to

the next node. Constraint (3.4) ensures that the destination must have one incoming virtual link for each file. Constraint (3.5) ensures that there is no splitting in the path for a particular file. Constraint (3.6) is flow-constraint equation for balanced flows. The number of incoming virtual links at a node for a particular file should equal the number of outgoing virtual links for that file.

2) *No time-overlap constraints*: These constraints ensure that, if a link is utilized for transferring one file, then it can be used for another file only after or before the file has been transmitted, but not during. At least one of the following three constraints must be satisfied.

For any link (*i, j*) and pair of files (*m,m_*):

$$V_{i,j}^{m} + V_{i,j}^{m"} \leq 1 \quad \ldots\ldots (3.7)$$

$$\tau(m") \geq \tau(m) + T(m) \quad \ldots\ldots (3.8)$$

$$\tau(m) \geq \tau(m) + \mathrm{T(m)} \quad \ldots\ldots (3.9)$$

Constraint (3.7) implies that link (*i, j*) is not shared by the two files (*m,m_*). Constraint (3.8) implies that link (*i, j*) is used for transferring file *m_* only after file *m* has been transferred. Constraint (3.9) implies that link (*i, j*) is used for transferring file *m* only after file *m_* has been transferred.

1) *Subject variable constraints*:

$$\tau(m) \geq 0 \quad \forall m \in M \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots (3.10)$$

Objective function:

The objective function of the proposed lambda grid scheduler is aimed at minimizing the time at which the last file is received at the destination (i.e. the *finish time)*

$$Minimize\left(Max\big(\tau(m) + T(m)\big)\right) \quad \forall m \in M \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots (3.11)$$

For the Lower bound on finish time, let the number of optical links through which destination *d* is connected to its neighboring switch nodes be *l*. Since only one file may be transferred along a link at a time, *d* may receive only *l* files simultaneously.

Thus, a lower bound on the *finish time* may be stated as:

$$T_{f_{in}}^{lb} = \frac{\sum T_f}{l} \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots .. \ldots (3.12)$$

3.6.3 Wavelength Allocation Scheme for the Proposed System.

The design uses a varying bandwidth technique in the reservation of lambdas for file aggregation task in the lambda grid. This allocation scheme is specified in algorithm 1.

The problem is formulated as follows:

End host applications request lambdas for file transfers by specifying a three-tuple $(S_f^i$, $R_{T_{max}}^i, T_{(m)}^i)$ where S_f^i is the file size of the request, $R_{T_{max}}^i$ is a maximum rate limit for this request, and $T_{(m)}^i$, is the desired start time for the request. $R_{T_{max}}^i$ Requested rate (also means number of lambdas). This request emanates from the grid application to which the scheduler will have to allocate transmission resources (lambdas). The scheduler is to allocate varying amount of bandwidth (varying numbers of lambdas). This varying lambdas is represented as a vector. A varying allocation is characterized as follows: $\{(B_k^i, E_k^i, C_k^i$,**),k=1,…**B_k^i**},** where B_k^i is the start of the K_{th} time range, E_k^i is the end of the K_{th} time range, while C_k^i is the capacity allocated for the transfer in the K_{th} time range. The reason for making such a varying allocation is that it allows the scheduler to backfill any holes left in wavelength allocations from other requests (by reallocating idle lambdas).

The scheduler maintains available capacity function $\gamma(t)$.

$\gamma(t)$ = Capacity availability function: Total number of available lambdas at time t.

$\gamma(t)$ is expressed in the following form:

$$\begin{cases} m_z & P_z \le t < P_{z+1} \\ m & t \ge P_{Z_{max}} \end{cases} \text{ where } m_z \le m \text{ and } z = 1,2,\ldots..max$$

By keeping a count of the allocations of all scheduled light paths (lambdas), the scheduler knows when and how much link capacity is available for a new request. A file transfer request specifies $(\boldsymbol{S_f^i}, \boldsymbol{R_{T_{max}}^i}, \boldsymbol{T_{(m)}})$ .The scheduled resource is $\{(B_k^I, E_k^i, C_k^i), k = 1, \ldots\ldots.\tau^i\}$which is an allocation of capacity for different time ranges for request. Capacity allocation for a new request is made on a round-by-round basis, where a round consists of the procedures used to allocate capacity for a time range that extends between two consecutive change points in $\boldsymbol{\gamma(t)}$. In a time range between two consecutive change points, the scheduler determines whether the entire remaining file can be transferred or the holding time ends within this time range, and whether the available capacity is greater than or less than/equals to $\boldsymbol{R_{T_{max}}^i}$. The design specifies four cases corresponding to the four possible outcomes of these two decisions. At the end of each round, the scheduler computes the remaining size of the file and starts the next round.

ALGORITHM 1: Wavelength Allocation Scheme for the Proposed Scheduler.

Start algorithm:

Set time $\upsilon \leftarrow T_{(m)}^i$, and remaining file size

$\phi \leftarrow S_f^i \; k \leftarrow 1$

Repeat loop (start next round):

Find z such that $P_z \leq v < P_{Z+1}$ in the capacity availability function $\gamma(t)$

If $\gamma(v) = 0$, then reset $v \leftarrow P_{x+1}$ continue

Repeat loop (start next round):

Case 1: Number of available channels is less than/equal to $R^i_{T_{max}}$ and the whole file can be transmitted before the next change in the available capacity curve, i.e.

$\gamma(v) \leq R^i_{T_{max}}$, and **(**$(P_{z+1} - v)\gamma(v)\chi \geq \Phi$ **,** then

Set

$B^i_k \leftarrow v$,

$E^i_k \leftarrow v + \emptyset/(\gamma(v)\chi)\Phi$

$C^i_k \leftarrow \gamma(v)$

(the begin time, end time and capacity allocation for the K_{th} range of file transfer *i*). Set

$\tau^i \leftarrow k$ **(Total number of time ranges allocated to file transfer *i*)**

Terminate

repeat loop.

Case 2: Number of available channels is less than/equal to $R^i_{T_{max}}$ but the whole file cannot be transmitted before the next change in the available capacity curve, i.e.

$\gamma(v) \leq R^i_{T_{max}}$, and $(P_{z+1} - v)\chi \geq \phi$, then

set

$B^i_k \leftarrow \boldsymbol{v}$

$E^i_k \leftarrow \boldsymbol{v} + \phi/(\boldsymbol{\gamma(v)\chi}$

$C^i_k \leftarrow \boldsymbol{\gamma(v)}$

set

$k \leftarrow k + 1$

$v \leftarrow P_{z+1}$ and

$\phi \leftarrow \phi - (P_{z+1} - v)\chi\gamma(v)$ continue

Repeat loop (start next round):

Case 3: Number of available channels is greater than $R^i_{T_{max}}$ and the whole file can be transmitted before the next change in the available capacity curve, i.e.

$\gamma(v) > R^i_{max}$, and $(P_{z+1} - v)R^i_{max}\chi \geq \phi$, then

Set

$B^i_k \leftarrow v,$

$E^i_k \leftarrow v + \phi/(R^i_{max}\chi),$

$C^i_k \leftarrow R^i_{max}$

.Set

$\tau^i \leftarrow k$

Terminate

repeat loop.

Case 4: Number of available channels is greater than $R^i_{T_{max}}$, and the whole file cannot be transmitted before the next change in the available capacity curve, i.e.,

$\gamma(v) > R^i_{max}$, and

$(P_{z+1} - v)R^i_{max}\chi < \phi$, then

Set

$B^i_k \leftarrow v$

$E^i_k \leftarrow P_{Z+1}$

$C^i_k \leftarrow R^i_{max}$

Set

$k \leftarrow k + 1$

$v \leftarrow P_{Z+1}$, and

$\phi \leftarrow \phi - (P_{z+1} - v)\chi R^i_{max}$ continue

Repeat loop (start next round):

End repeat loop

$\gamma(t)$ is expressed in the following form:

$$\begin{cases} m_z & P_Z \leq P_{z+1} \\ m & t \geq P_{z_{max}} \end{cases} \quad \text{where } m_z \leq m, and\ z = 1,2,\ldots\ldots, z_{max}, \quad .$$

z_{max} denotes the number of Times $\gamma(t)$ changes value before reaching (*m)* at $t = P_{z_{max}}$ after which all (*m)* channels of the link remain available.

3.6.4 Path Determination Technique for the Proposed System

ALGORITHM 2

Consider a Network Topology Graph G (V,∈,W) where V is the set of nodes, ∈ is the set of links and W is the set of wavelengths(lambdas) supported by each link. A grid application file transfer request (which entails scheduling light paths (lambdas) between any two nodes on G).

As specified earlier, each host application on the grid request lambdas for file transfer by specifying three tuples: (S_f^i, $R_{T_{max}}^i$, $T_{(m)}^i$)**.** Adding extra information, this request is further expanded as:

Request = [source, destination, $\boldsymbol{T_{(m)}^i}$**, e, d,** $\boldsymbol{R_{T_{max}}^i}$] where **d** is reservation duration. This is estimated using equation (1), $\boldsymbol{T_{(m)}^i}$ and **e** are the starting and ending time of the schedule window respectively as illustrated in figure.3.5

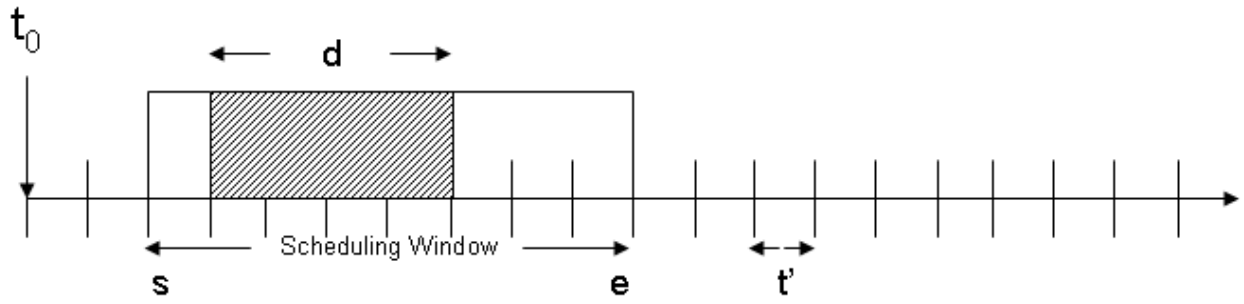

Figure3.7 Scheduling Window

The end time of the Scheduling window is estimated using equation (3.12). The time is slotted with a slot size equal to t. The scheduling window defines the time period within which the resource reservation is made. The scheduling window must be bigger than the reservation duration d. Thus the scheduler must check if a path is available during interval [$T^i_{(m)}$ + t, $T^i_{(m)}$ + t + d] where t = 0, 1, 2, ..., e - $T^i_{(m)}$ - d.

Recall that the lambda grid resource scheduling problem is formulated as TPSP. The path determination scheme used here is that when a request arrives, the scheduler first try to find the shortest available path starting at time $T^i_{(m)}$ for d slots. This is done by first finding the shortest path, using Dijkstra's algorithm. Then, the scheduler checks if all the links on this path have a free wavelength for d slots starting at time $T^i_{(m)} + t$, where t=0. If any link is busy along the path, the topology is updated by removing that link and the next shortest path is determined. This step is repeated until either a path is available or a

maximum of k different paths have been considered. If a path cannot be determined, the scheduler repeats the whole process with a start time equal to $T^{i}_{(m)} + t$, where t=1. t is incremented by one slot each time until an available path is found or t = e $- T^{i}_{(m)} -$ d, whereupon a request is blocked.

NOTE: in checking for links and determining path, the algorithm imposes the constraints defined earlier in this chapter (constraints (3.2),(3.3),(3.4),(3.5)) on the schedule

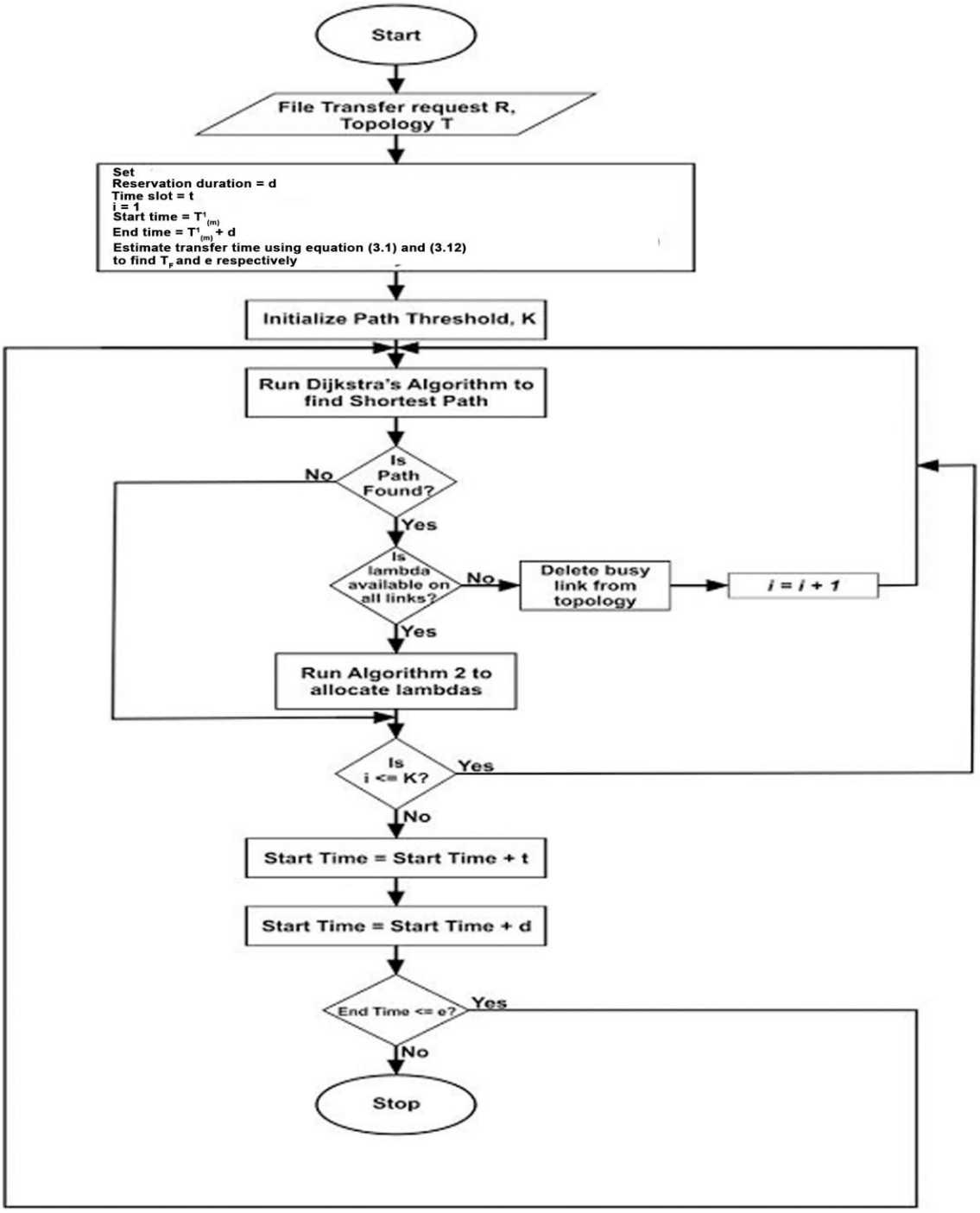

Figure 3.8 Flow Chart for the Path Determination Algorithm

3.6.5 The Proposed Scheduler

ALGORITHM 3

The algorithm sorts all file transmission tasks (see figure 3.4) into a scheduling list LIST according to certain priority scheme. The route (path) of a file transmission is determined by algorithm 2. The algorithm iterates over the list. An unscheduled task is selected from the LIST with the highest priority, then allocate optical link resources which builds up its route for time interval equal to f_n's transmission time (as determined using equation (3.1). The priority scheme used in the algorithm is the longer file first (LFF). The LFF is a heuristic that is based on the intuition that the longest file (having the largest transfer time) is the bottleneck for scheduling, because it requires more resources in terms of the amount of time required to be free on the links to be transferred. The LFF algorithm aims at scheduling the longest files first so that they get the priority to be scheduled earlier.

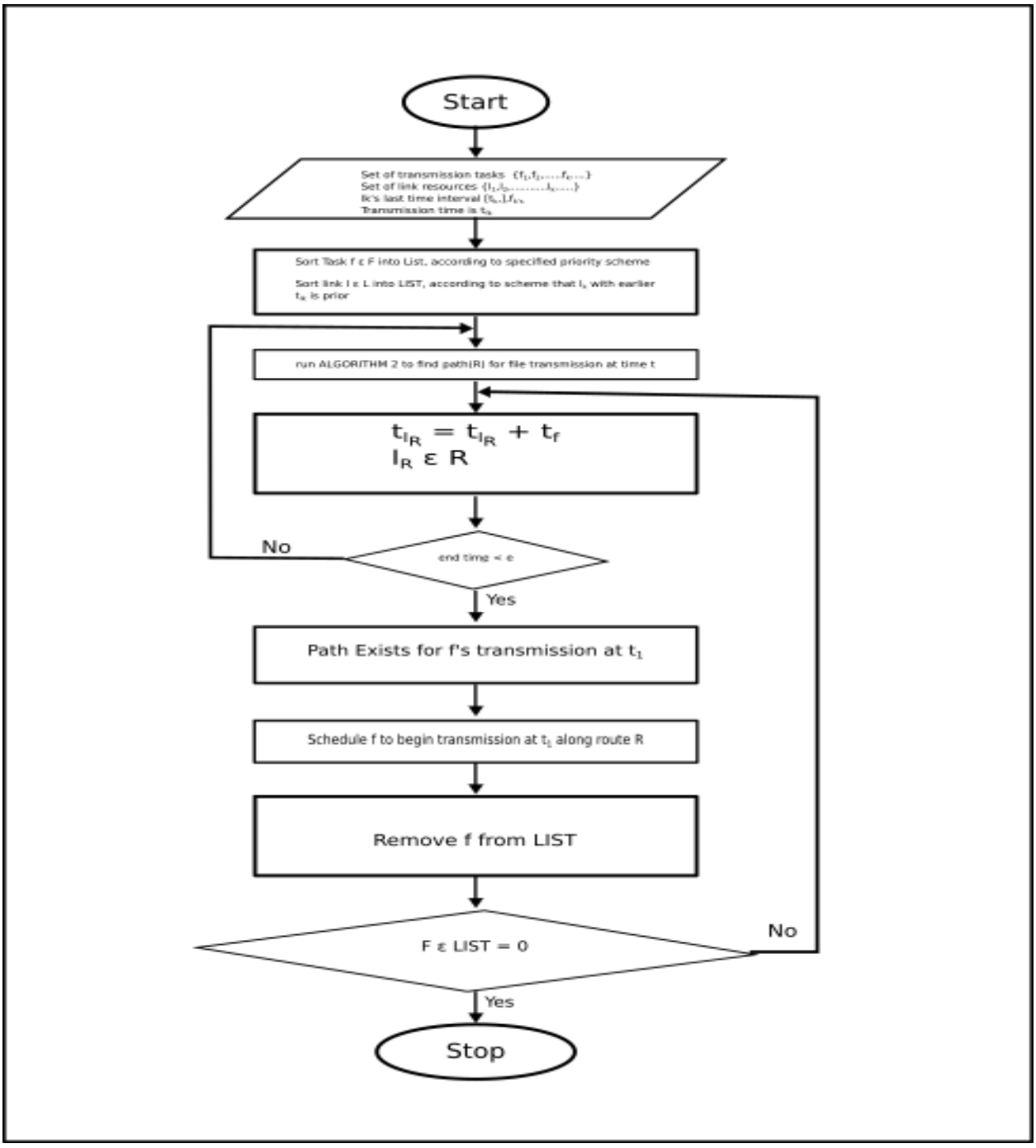

Figure 3.9 Scheduler flow chart**.**

CHAPTER FOUR

SIMULATION AND RESULT ANALYSIS

4.1 Chapter Overview

In this chapter simulation is setup and carried out to test and evaluates the proposed lambda grid scheduling algorithm. The case study (sample) lambda grid topology modeled for simulation is the 24-node National Lambda Rail (NLR). The NLR is lambda grid network owned and operated by the U.S. Research and education community (Travostino, F. et.al., 2006). The NLR is primarily oriented to aid tera scale computing efforts and is used as a network test bed for experimentation with large-scale networks. NLR aims to enhance system-level integration of new technologies that cut across traditionally defined network layers and enable critical applications like resource management (such as the lambda grid scheduling algorithm proposed in this work) and security. In terms of traffic measurement and data analysis, NLR provides the network research community with visibility and access to underlying transmission, switching and routing fabrics in a way that is not possible to achieve in pure production or commercial environment (Guerin, RA, et.al., 2000).

The foundation of NLR is the dense wave division multiplexing (DWDM) based national optical footprint using Cisco systems 15808 optical electronic system with a capacity of 40channel (wavelength) per fiber pair. Each wavelength can support transmission at 10 billion bits per second! (10Gbps).

The single line diagram of the NLR 24-node lambda grid sample network topology used in the simulation carried out in this chapter is given in Figure 4.1.

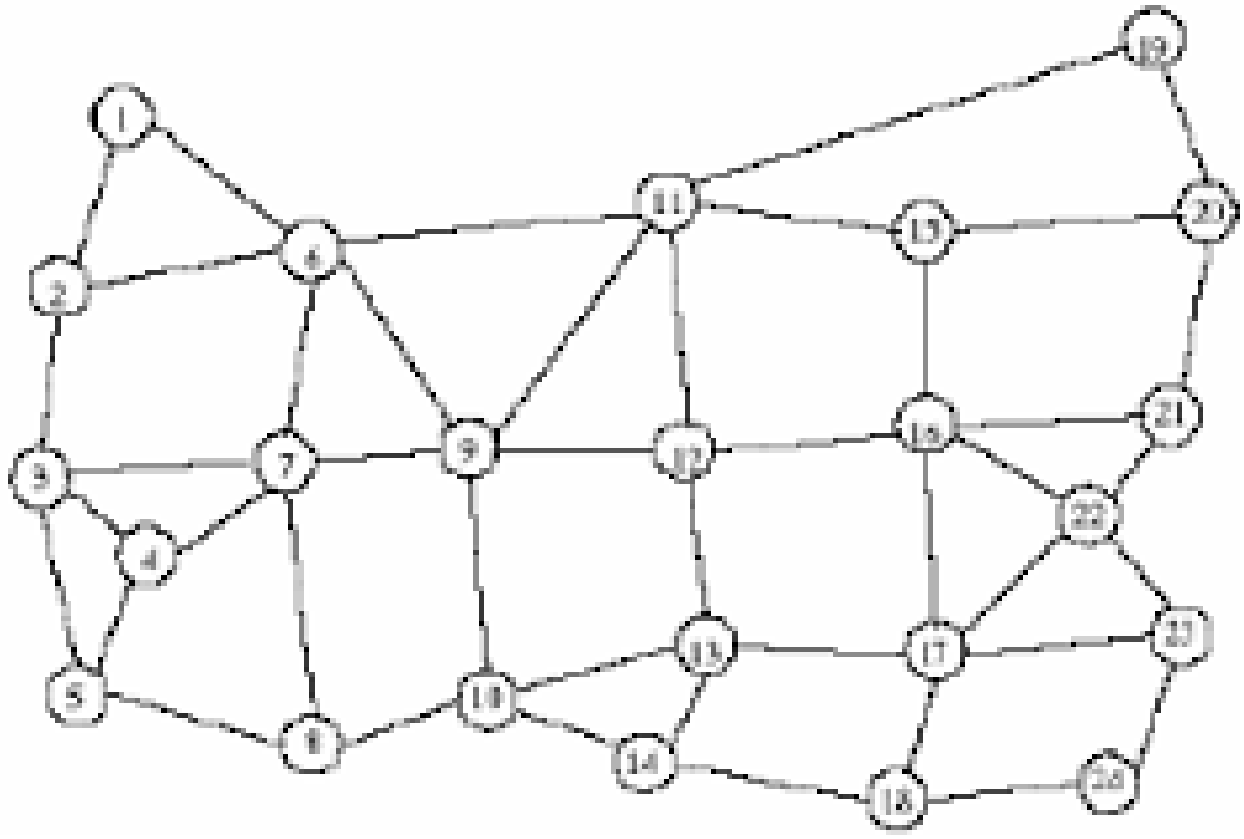

Figure 4.1 Single line diagram of the NLR 24-node lambda grid sample network topology used in the simulation.

The Cisco packet tracer 7.1 (Wang Xia-hong., 2012). is used to create the digital model of the 24-node lambda grid network topology. This is shown in figure 4.2.

The nodes in figure 4.2 are multilayer switches. These nodes perform layers 2, 3and other upper layer networks functions. The lines between nodes are 0C-192, while that between node and super computers are 0C-48 specification.

The Cisco network simulator has application programming interface (API) support for the C++, java and C programming languages. For this work the C++ programming language was used. The proposed lambda grid scheduler is coded in the C++ language. It interfaces and communicates with the kernel of the Cisco packet tracer network simulator using inter procedural communication based on the CORBA protocol (common object request broker architecture). C++ code is used to automate the scripting of file transfer and file aggregation jobs between nodes in the network during the simulation run. The source code is given on appendix C. and the Simulation processing trace file is given on appendix D.

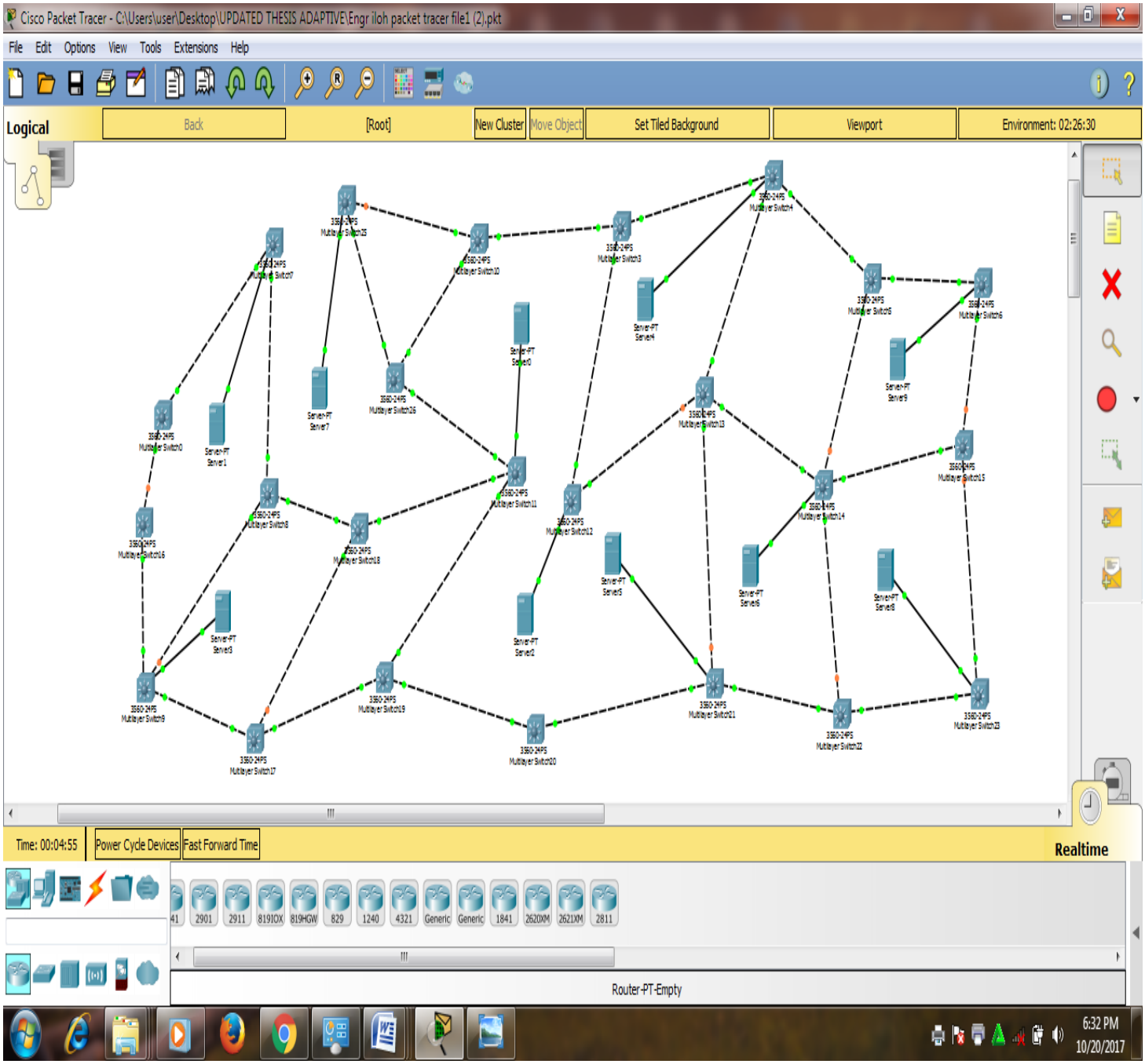

Figure 4.2: Model of the 24-Node Lambda Grid network topology for evaluating the performance of the proposed adaptive lambda Grid Scheduling Algorithm

4.2 File Transfer Time

In the simulation, the program code increases the number of files gradually from 50 to 300. For each setting, the program measures the file transfer finish time. All link capacities are C= O C-192 (10 Gigabytes). Source and destination node where the super computer exists are automatically selected by the program. A specified number of files of sizes are randomly distributed between 5 Gigabytes and 20 Gigabytes and are located randomly across and the remaining nodes in the lambda grid network.

Figure 4.3 compares the performance of the proposed adaptive lambda gird scheduling algorithm; the Varying Bandwidth List Scheduler (VBLS) and the Virtual Finish time (ViFi) grid scheduler. It can be observed from figure4.3 that the proposed adaptive scheduler performs better than the VBLS and the ViFi algorithms. The VBLS algorithm performs better than the ViFi algorithm. Furthermore, it can be noticed that the difference in performance gets even more distinct as the number of files increases. In other words, with increasing number of files, the performance of the proposed adaptive algorithm increases and the margin with which it outperforms the other two algorithms widens. This is to be expected: that as the number of files to be transferred between nodes increases, the algorithm adapts by rerouting data and **re-provisioning** idle lambdas (i.e wavelengths) in order to **effectively service the additional work loads**. This gives the proposed algorithms an edge over the other two algorithms. The finish time for transferring same amount of data **is less** for the proposed adaptive scheduler compared to the other two schedulers.

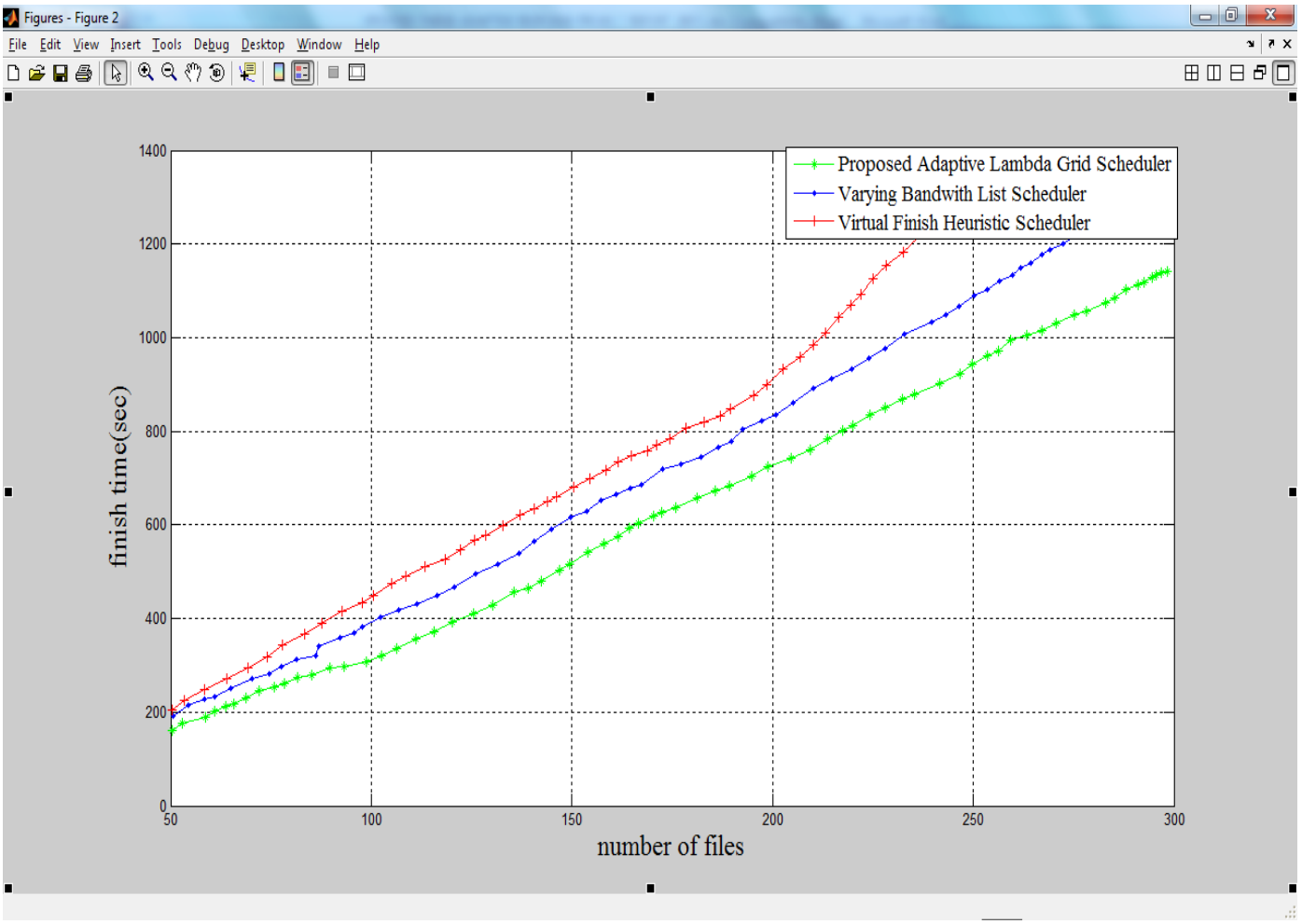

Figure 4.3: Comparison of the File Transfer Finish Time of the Proposed Adaptive Lambda Grid Scheduler, the Varying Bandwidth List Scheduler and the Virtual Finish Scheduler

To compute and compare averages for the three schedulers, ten readings are taken off the graph of figure 4.3. The readings are taken at intervals of 25 along the horizontal axis (number of files) starting from 50. The outcome is tabulated in table 4.1.

Table 4.1 File Size versus Finish Time for File Aggregation using the Proposed Adaptive Schedule, the various Bandwidth List Schedule and the Virtual Finish Heuristic Scheduling Algorithms

	Finish time (sec.).		
	Proposed Adaptive Gird Schedule Algorithm	VBLS algorithm	ViFi algorithm
50	161.80	189.40	199.50
75	249.60	284.80	322.40
100	307.30	390.10	447.80
125	397.70	485.50	555.70
150	513.10	615.50	678.70
175	628.50	718.80	922.00
200	723.80	829.20	1125.30
225	836.00	944.60	1300.90
250	944.60	1077.60	1353.60
275	1037.50	1208.10	1376.30

.

From the tabulation, the average finish time for the aggregation of the files randomly distributed between 5 Gigabytes and 20 Gigabytes is computed.

From the tabulation, the average finish time for the file aggregation using the proposed adaptive lambda grid scheduling algorithm, the VBLS scheduling algorithm and the Virtual Finish (ViFi) algorithm are **579.99** seconds, **674.36** seconds and **828.22** seconds respectively. With this average, the proposed adaptive lambda grid scheduler provides;

For % improvement on Varying Bandwidth List Scheduling (VBLS);

674.36 -579.99=94.37, then % improvement becomes 94.37/674.36*100/1=13.999%.

For % improvement on Virtual Finish (ViFi);

828.22-579.99=248.23, then % improvement becomes 248.23/828.22*100/1=29.97%.

approximately 14% and 30% improvements in grid file aggregation finish over the VBLS algorithm and the virtual finish algorithm respectively.

4.3 Evaluation of Effect of Late Arrival Rate on Blocking Probability

The rate of arrival of request for allocation of optical wavelengths (lambdas) has impact on the job blocking probability (i.e. the probability of not scheduling a request within its window). As this is an online scheduling problem, the request arrives dynamically and

for each request, the scheduling algorithms must compute a path (routing) and then check if a wavelength (i.e. a lambda) on each link of this path can be reserved for a duration within the scheduling window (the scheduling window is specified by the schedule **start time** and **end time**). The scheduling algorithm allocates a lambda on each link along a path from the source to the destination nodes. If a lambda along the path for the specified period of time is not available, another path has to be determined.

The objective of the algorithm is **to determine the schedule** to route each incoming light-path connection request dynamically while **minimizing the probability** that a connection request will be refused due to lack of available light path and maximizing the overall network throughout. Figure 4.4 shows the effect of the arrival blocking probability.

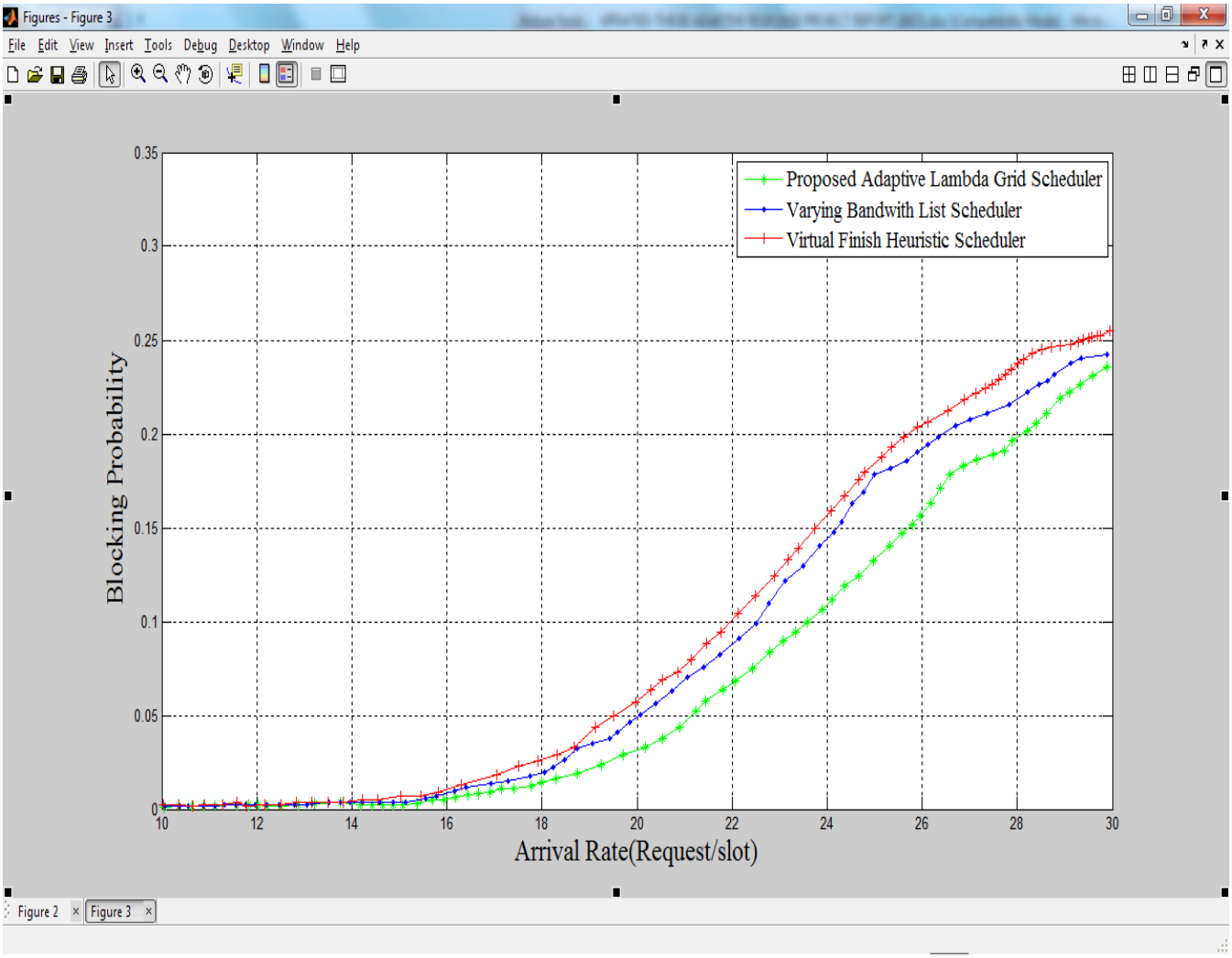

Figure 4.4: Effect of lambda grid request arrival rate on blocking probability

The general observation from the graph is that blocking probability increases with increase in request arrival rate. It seems to increase exponentially. It can be observed for the three algorithms that the probability of not scheduling a lambda request within its window is very infinitesimal (almost zero in this case) for arrival rate below 14 request/slot. With low values of arrival rate, the blocking probabilities of the three

algorithms seemed almost equal. However, with increasing arrival rates the difference in the performances of the algorithms in terms of blocking probabilities begin to significantly stand out.

Even more prominent is the significant difference in the blocking probabilities of the proposed adaptive scheduler with those of the VBLS and ViFi. This means that, at increasing arrival rate, for the algorithm to meet its objective of minimizing blocking probabilities it has to bring in adaptability.

To compute and compare averages of blocking probabilities of the three algorithms, readings are taken off the graph of figure 4.4 at 10, 16, 22 and 28 along the horizontal axis. The outcome is given in table 4.2.

Table 4.2: Tabulation of Request/Slot for Allocation of Optical Wavelength against the Blocking Probabilities of the Proposed Adaptive Scheduling Algorithm, the VBLS Algorithm and the ViFi Algorithm.

Request/slot	Blocking probability		
	Proposed Adaptive Grid Scheduling Algorithm	VBLS Algorithm	ViFi Algorithm
10	0.0007	0.0007	0.0012
16	0.0047	0.0078	0.0100
22	0.0658	0.0886	0.1007
28	0.1958	0.2185	0.2355

Based on table 4.2, the average blocking probabilities for the proposed scheduler, the VBLS scheduler and the ViFi scheduler are 0.0667, 0.0789 and 0.0866 respectively. It is clear the proposed adaptive scheduler was the lowest blocking probability.

For % improvement on VBLS:

0.0789-0.0667=0.0122, 0.0122/0.0789*100/1=15.4%

For % improvement on ViFi:

0.0866-0.0667=0.0199, then % improvement is 0.0199/0.0866*100/1=22.97%

This value indicates a 15.4% and 23% improvement over the VBLS algorithm and the ViFi algorithm respectively

4.4 Evaluation of Effect of Connection Functions on Blocking Probability.

Figure 4.5 gives the effect of the connection duration **d** on the blocking probability. It can be seen that the blocking probability increases as connection duration increases. It can also be observed that the blocking probability significantly increases for connection duration >5 for the considered topology. As can be observed the proposed adaptive scheduler gives the best performance of the three, followed by the VIBLS algorithm.

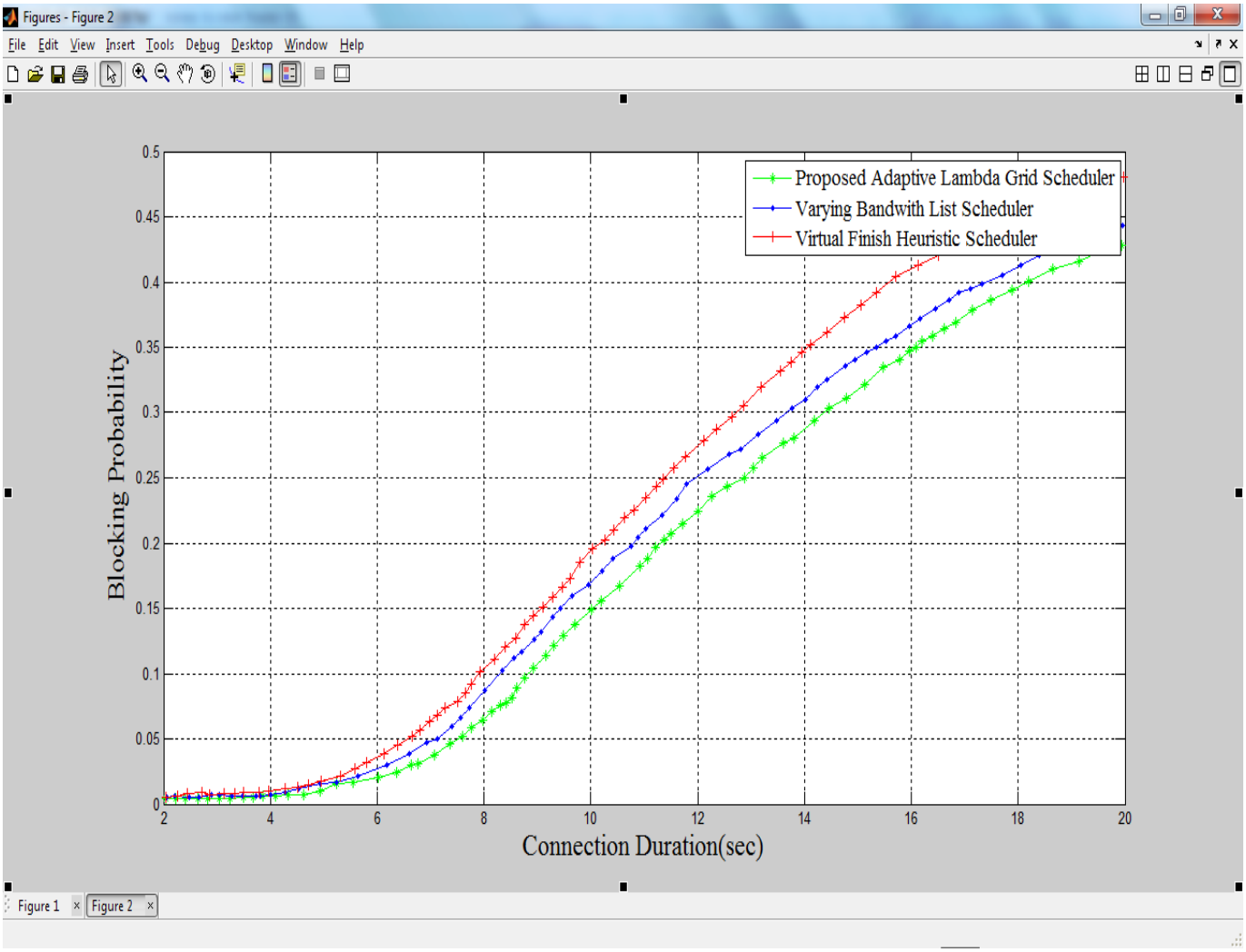

Figure 4.5: Connection Duration Verses Blocking Probability.

Table 4.3 is setup by taking readings at location 6, 10, 14 and 18 along the horizontal axis in figure 4.5

Table 4.3: Comparison of the Effect of Connection Duration on Blocking Probability on File Aggregation for the Proposed Adaptive Scheduler, the Virtual Bandwidth List Scheduler and the Virtual Finish Heuristic Scheduling Algorithms.

Connection duration (sec)	**Blocking probability**		
	Proposed Adaptive Gird Scheduling Algorithm	VBLS Algorithm	ViFi Algorithm
6	0.0174	0.0228	0.0371
10	0.1469	0.1710	0.1960
14	0.2879	0.3121	0.3478
18	0.3969	0.4121	0.4549

Based on table 4.3, the average blocking probabilities as a result of the impact of connection duration are **0.2122, 0.2295** and **0.2589** for the proposed adaptive scheduler, the VBLS and the ViFi algorithms respectively. These figures indicate that the proposed algorithm has the lowest blocking probability.

For % improvement on VBLS:

0.2295-0.2122=0.0173, then % improvement; 0.0173/0.2295*100/1= 7.5%

For % improvement on ViFi:

0.2589-0.2122=0.0467, then % improvement; 0.0467/.2589*100/1= 18%

These values represent a **7.5%** and **18%** improvement over the VBLS algorithm and the ViFi algorithm respectively.

4.5 Evaluation of light-path Reservation Delay as a Function of Wavelength Request Arrival Rate.

Figure 4.6 shows the reservation delay, i.e. the time elapsed from the requested start time s to the time equal to the start time plus the time slot, as a function of lambda request arrival rate for the proposed adaptive algorithm, the VBLS algorithm and the ViFi algorithms.

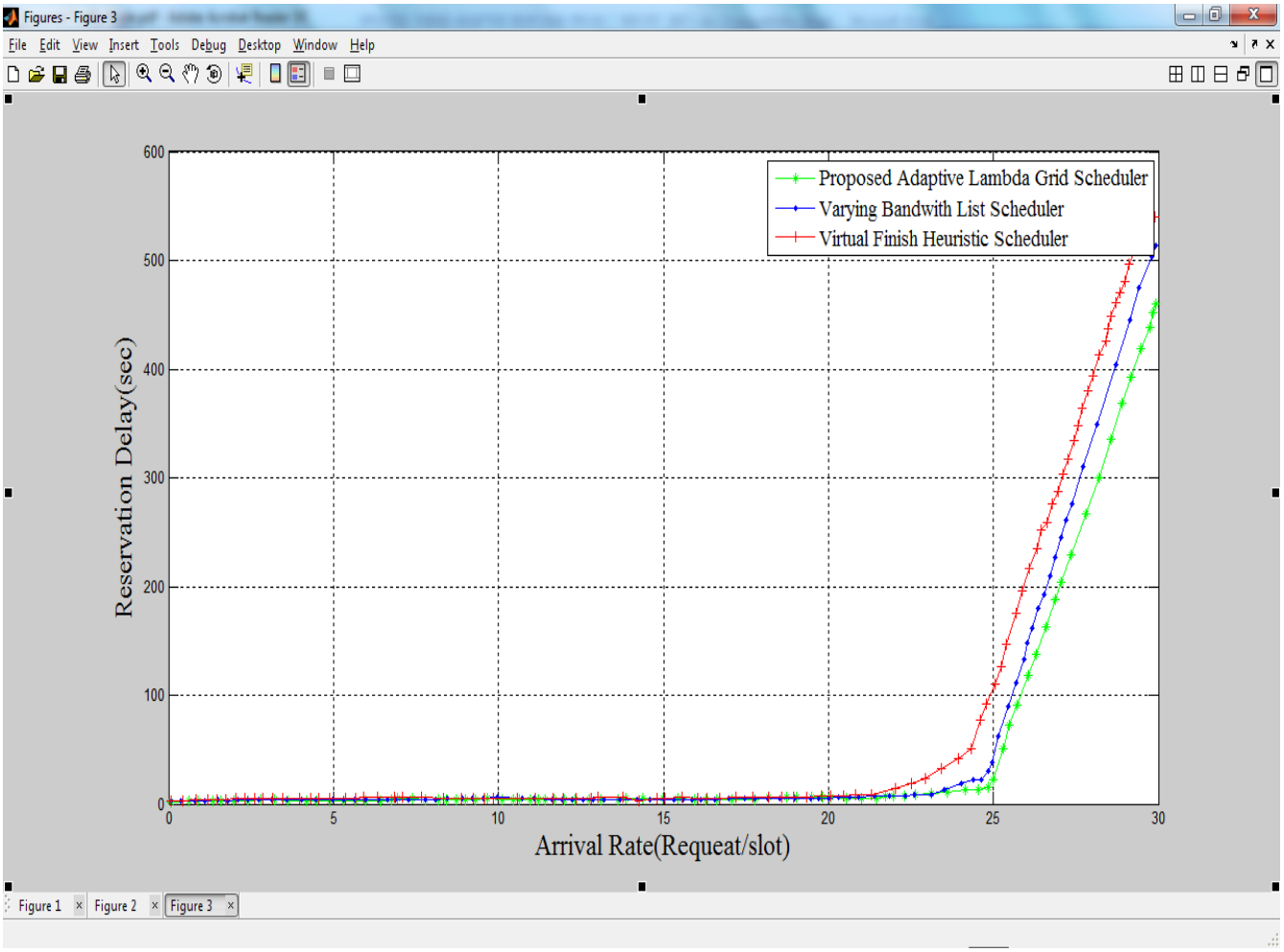

Figure 4.6: Reservation Delay as a function of wavelength Request Arrival Rate

It can be observed that for arrival rates below the threshold point of 20, the reservation delay algorithms are almost equal. Then around 25 request/slot of the reservation delay for each of the algorithms rises rapidly. It can be seen that based on the reservation; the proposed algorithms stand out from the other two algorithms. In comparison, it gives the least reservation delay as per impact of request for lambda arrival rate. This means that the proposed adaptive algorithm always tries to schedule close to the start time of the scheduling window as possible.

4.6 Evaluation of Job Blocking Rate

The numbers of jobs simulated are varied up to 200. The number of tasks per job is varied in the program code up to 6. The job size was determined in program code based on the number of jobs that were submitted to the scheduler. Figure 4.7 shows the job blocking rates of the three grid scheduling algorithms. The job blocking rate is the percentage of jobs blocked divided by the total number of jobs submitted.

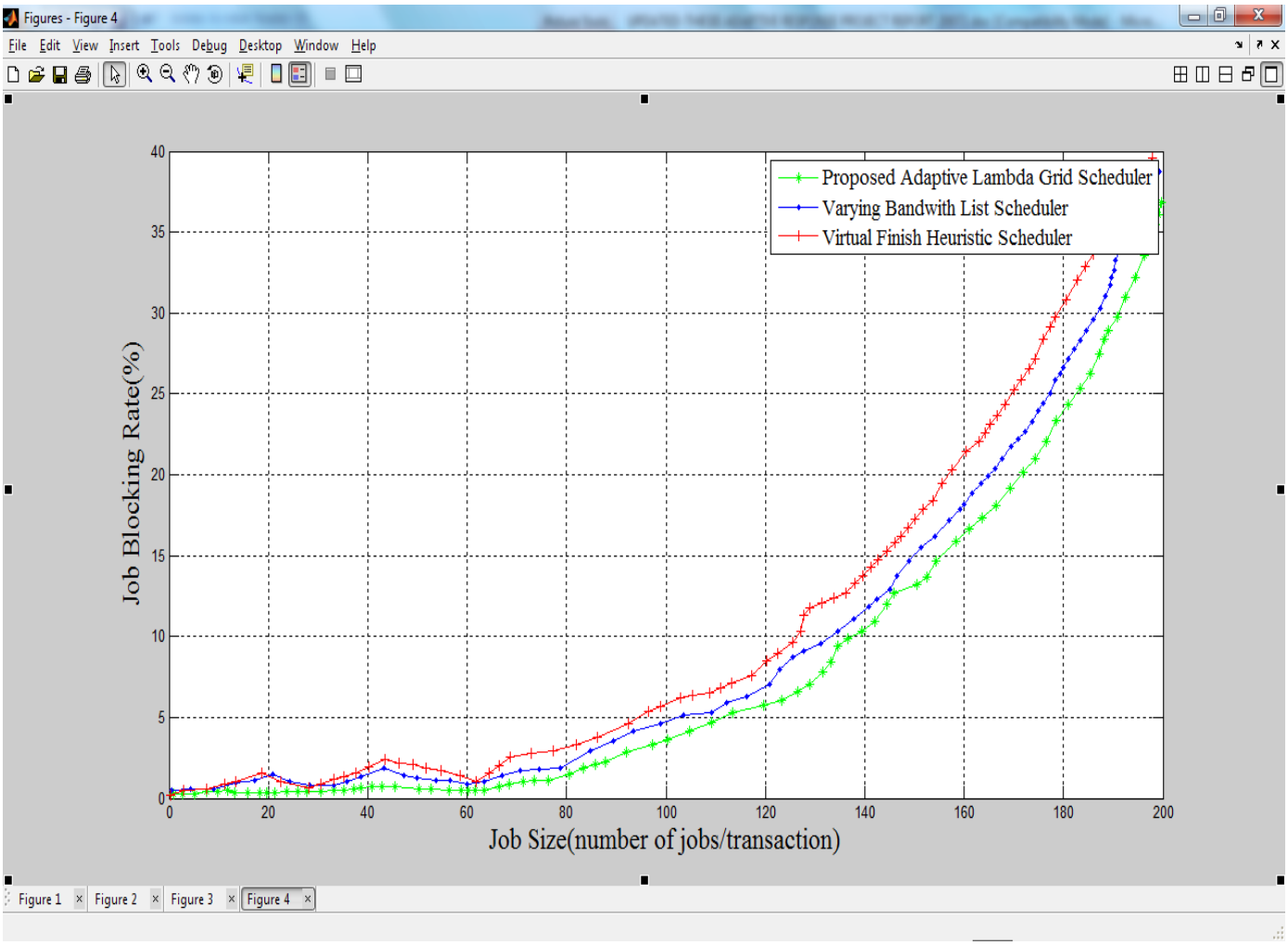

Figure 4.7: Average Job Blocking Rate

As can be observed, the proposed adaptive scheduler clearly out performs the other two algorithms. As can be observed, the job rates using the VBLS and ViFi algorithms are more than that using the proposed algorithm. That is, the proposed algorithm has minimal blocking rate compared to the other two algorithms. It is also evident from figure 4.7 that the blocking rate of all three algorithms increases dramatically with the increase in job size. However, from the result; it is evident that the proposed algorithm reduces the blocking rate in comparison. Furthermore, it can be observed from figure 4.7 that the job blocking rate using the proposed adaptive algorithm does not vary and oscillate as those of the VBLS and ViFi algorithms.

To estimate the average for job blocking rate, readings are taken off the graph at 20, 60, 100, 140 and 180 positions along the horizontal axis. Table 4.4 gives the readings for the three algorithms.

Table 4.4: Job Blocking Rate versus Job Size.

Job size	Job blocking rate		
	Proposed adaptive grid scheduling algorithm	VBLS algorithm	ViFi algorithm
20	0.1786	0.9643	1.4643
60	0.2500	0.8214	1.3214
100	3. 6071	4.6786	5. 7500
140	10.2500	11.8214	14.0357
180	23.3643	20.8214	30.6071

From the tabulation, the average job blocking rates is calculated using the formula:

Job blocking rate = % of jobs blocked /Total nos. of jobs submitted

are **7.53%, 9.02%,** and **10.64%** for the proposed adaptive algorithm, the VBLS algorithms and the ViFi algorithms respectively.

Then, % improvement on VBLS regarding job blocking rate:

9.02-7.53=1.49; 1.49/9.02*100/1= 16%

For % improvement on ViFi regarding job blocking rate:

10.64-7.53=3.11; 3.11/10.64*100/1= 29%

The computed average for the proposed adaptive scheduling algorithm represents a **16%** and **29%** improvement over the VBLS and the ViFi algorithms respectively.

4.7 Evaluation of the Effectiveness of the Scheduling Algorithms

The effectiveness is calculated as the percentage of latest finish time of the job scheduled and the blocking rate to the maximum time slots. The higher the percentage, the more effective the algorithm.

The algorithm computes the effectiveness using equation (2.1)

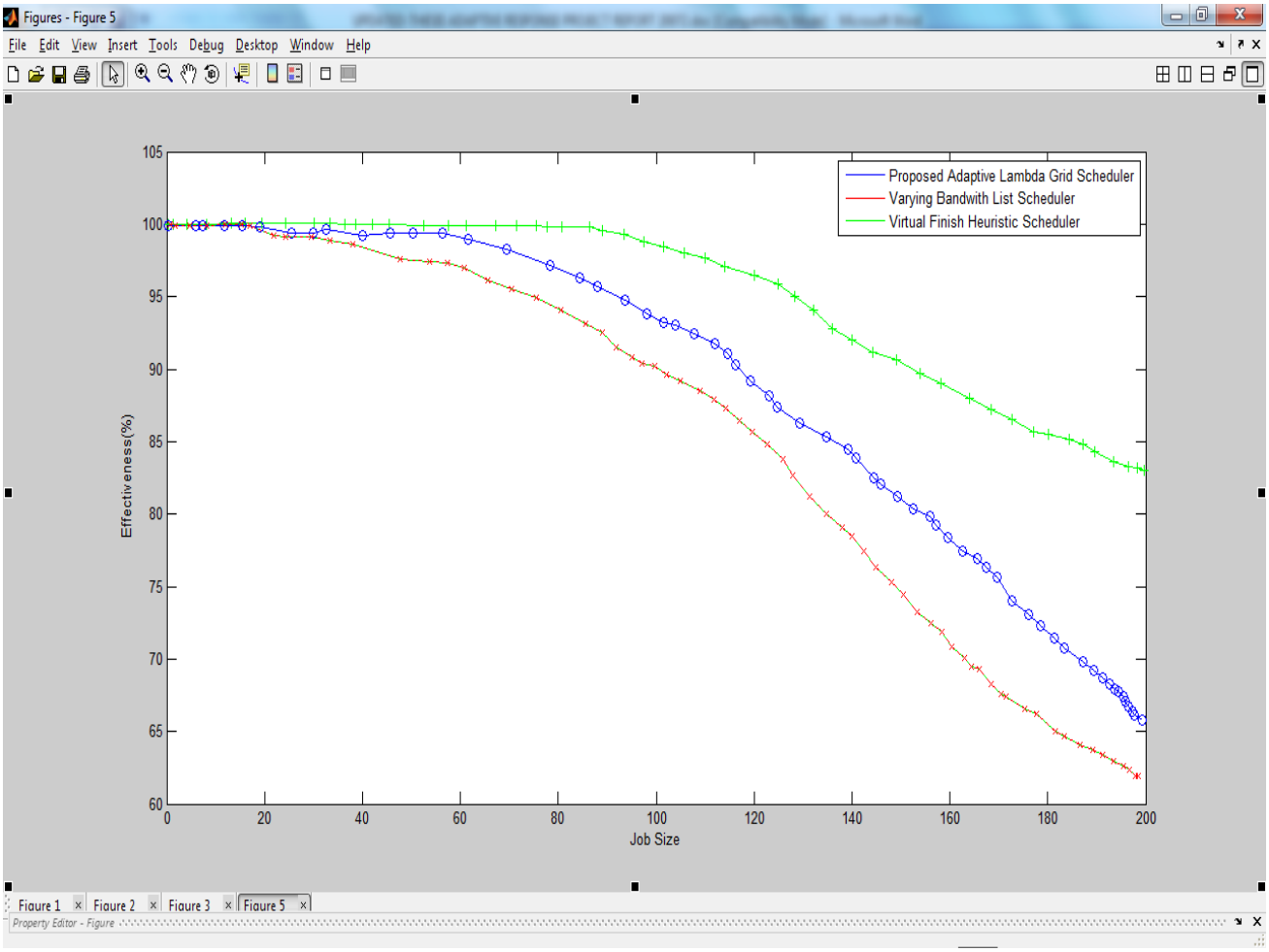

Figure 4.8: Shows the variation of effectiveness with job size for the three algorithms.

It can be observed from the graph that effectiveness reduces with increase in job size. The simulation result indicates that the proposed adaptive scheduling algorithm has the best effectiveness of the three algorithms. As can be seen the effectiveness of the algorithms reduced from almost **100%** to about **83%, 65%** and **61%** from the graph for the proposed adaptive algorithm, the VBLS and the ViFi algorithms respectively. The higher the %, the more effective the proposed algorithm becomes.

Then, the improvement of the algorithm against VBLS is calculated as;

83% - 65% = 18%

83% - 61% = 21%

It can be inferred from this that the proposed adaptive algorithm performs 18% better than the VBLS algorithm and 21% better than the VIFI heuristic grid scheduling algorithm respectively.

CHAPTER FIVE

CONCLUSION, CONTRIBUTION TO KNOWLEDGE AND RECOMMENDATION

5.1 Conclusion

This dissertation focused on the development of an adaptive resource scheduling technique to minimize the delay in the data aggregation task in a computational lambda grid network.

The problem of data aggregation delay in the lambda grid has enormous impact on the viability of certain e-science application that have critical timing requirements, the loss of teraflops of super computer computing power as a result of the scheduling related delay in the lambda grid has the impact of increase in research and development (R & D) cost, delays in proceeding with vital research (especially related to chronic disease research etc.). This problem not only leads to delays in vital break through for mankind but also, in some cases outright project cancellation and wasted investment. The lambda grid scheduling problem has far reaching impact on the accuracy and validity of e-science grid applications required in making vital forecast relating to natural disaster, evolving disease postures, climate change, technical issues relating to our ability to explore deep space etc.

Grid computing emerged as a means of coupling together numerous heterogeneous and geographically distributed computational and storage resource to make them work as a unified resource. By coupling numerous heterogeneous computational and storage

resource distributed over various locations, Grids are able to satisfy the ever increasing demand of both processing and storage power, surpassing the capabilities of each of its individual resources. This allows a grid to accommodate even the largest and most resource- demanding applications. Grids making use of optical circuit switched transport network are usually denoted as lambda grids.

One is said to be pushing the current increase in the development of lambda grids networks to the resource intensive requirements of e-science application. Many of these data-intensive, e-science Gids applications like electronic very long Baseline interferometry (e-VLB) and Genomes to life (GTL) requires aggregating several hundred Gigabytes of data files from distributed databases (usually geographically separated) to computing resource (such as supercomputers) frequently in real time since data is aggregated at the time of computation, the time required to transfer the data over the network is the main computational bottleneck.

The design of the proposed adaptive lambda grid scheduling was carried out in this work. In this work the lambda grid scheduling problem is formulated as a time-path scheduling problem. The design carried out constructed the scheduler as a three algorithm system. Algorithm 1 is designed to allocate lambdas, the formation of algorithm 2 is to determine the file transmission path in the lambda grid, and Algorithm 3 implements the lambda grid resource scheduler. The scheduler is the module that schedules the actual file transfer. Every run of the path determination algorithm (Algorithm 2) is integrated with the running of the lambda allocation algorithm (i.e Algorithm 1). The scheduler

iteratively runs the path algorithm to dynamically re-establish the shortest path from the source to the destination (with the consequent re-allocation of wavelength). This re-allocation of wavelength ensures that idle lambdas can be re-provisioned for ongoing or later file transfer in the lambda grid. The proposed scheduler design is coded in the C++ programming language.

In the work, simulation was setup and carried out to test and evaluate the proposed lambda grid scheduling algorithm, for the required data, the 24-node National Lambda Rail (NLR) lambda grid topology was used. Cisco packet tracer for network modeling software was used to create the digital model of the 24-node lambda grid network topology. The Cisco network simulator with application programming interface (API) support for the C++, java and C programming languages.

In the simulation, the program code increases the number of file gradually from 50 to 30. For each setting, the program measures the file transfer finish time. All link capacity is OC-192 (10 Gigabytes). Source and destination node where the supercomputer exists are automatically selected by the program. A specified number of file size are randomly distributed between 5Gigabytes and 20Gigabytes and are located randomly across and the remaining node in the lambda grid network.

The performance of the proposed adaptive lambda grid scheduling algorithm was analogically compared with the varying Bandwidth List scheduler (VBLS) and the virtual finish (ViFi) grid scheduler. The observation is that the proposed adaptive scheduler

performs better than the VBLS and the ViFi algorithms. The VBLS algorithm performs better than the ViFi algorithm. The difference in performance gets more destruct as the number of file was increased as during the simulation carried out. With increasing number of file, the performance of the proposed adaptive algorithm increased and the margin with which it outperforms the other two algorithms widened. This is to be expected, that as the amount of data to be transfer between nodes in the lambda grid increases, the proposed algorithm adopts by searching data and re-provisioning idle lambdas (i.e. wavelength) in other to effectively service the additional work loads. This gives the proposed algorithm an edge over the other two algorithms.

The finish time for transferring same amount of data is less for the proposed adaptive scheduler compared to the two schedulers.

Result obtained by the evaluation of the average finish time for file aggregation showed that the proposed algorithm achieved 14% and 30% improvement over the VBLS algorithm and the ViFi algorithm respectively.

The rate of arrival on blocking probability (ie the probability of not scheduling a request within the window) was evaluated. Comparative analysis carried out show that the proposed scheduler has the lowest blocking probability. The blocking probability of the proposed algorithm, the VBLS algorithm and the ViFi algorithm are 0.0667, 0.0789 and 0.0866 respectively. This value showed that the proposed algorithm achieved a 15.4% and 22.9% improvement in blocking probability over the VBLS and ViFi algorithm

respectively. It was observed that the blocking probability increases with increase in request arrival rate. The increase seemed to be exponential. With low values of arrival rate the blocking probability of the algorithm seemed almost equal. However, with increasing arrival rates the difference in the performances of the algorithm in terms of blocking probability begins to significantly stand out.

The effect of connection duration on blocking probability was evaluated. Result obtained indicates that blocking probability increases as connection duration increases. The proposed algorithm has the least blocking probability with increase in connection duration. Numerical results show that the proposed algorithm achieved 7.5% and 18% improvement over the VBLS algorithm and the ViFi algorithm respectively.

Light-path reservation delay as a function of wavelength request arrival rate was evaluated. Result obtained indicates that the proposed algorithm gives the lowest reservation delay as per impact of request for lambda arrival rate. This means, of the three algorithms compared, the proposed adaptive algorithm always tries to schedule close to the start time of the scheduling windows as possible.

The proposed algorithm was found to have the least blocking rate in comparison with the other two algorithms. Furthermore, findings indicate that the job blocking rate of the proposed algorithm does not vary and oscillate as those of the VBLS and ViFi algorithm. The average blocking rates are 7.55%, 9.02% and 10.64% for the proposed algorithm, the VBLS and the ViFi algorithm respectively. These values show that the proposed

algorithms achieved a 16% and 29% improvement over the VBLS and the ViFi algorithm respectively.

Evaluation of the variations of effectiveness of the algorithm with job size was carried out. Finding show that the effectiveness reduces with increase in job size, Simulation results indicates that the proposed adaptive algorithm performs 18% better that the VBLS algorithm and 21% better than the ViFi heuristic grid scheduling algorithm.

5.2 Contribution to Knowledge

The main contribution of this work is the algorithm for the adaptation of the LIST scheduling algorithm to blend the Dijikistra algorithm and TPSP algorithm.

This scheme is found to improve on the large file first and to optimize the light path determinations computation.

5.3 Recommendation

It is of vital importance that a high end national optical transport is made available to Nigerians and international researchers for measurement, experimentation and business operation purposes. A key recommendation here is that a national lambda grid network (probably with the code name **NigerGrid**) should be constructed. Specific wave length should be allocated for intensive e-science project, specific computational grids, Tele-

presence or other scientific experiments. NigerGrid is to provide the real physical environment not only to move the algorithm proposed in this work to operational status, but to also enable research in to innovative optical transport technique and to aid high end, complex e-science research and enhance e- collaboration among Nigeria Universities and research centers across the country.

This project report recommends that the management of NigerGrid be constituted under a joint arrangement comprising Nigerian Universities, Research Centers and Nigeria Defense incorporation.

In the present work minimization of finish time is the main objective function in the design of the adaptive lambda grid scheduling algorithm, it is here recommended that further work should expand on the objective function to include minimization of transmission energy consumption and the compute cycles on the OC 192 core network node device during file aggregation.

References

Aanchal Bawa, Sonam Bhata and Varinder Kamr. (April,2015) Altr: "A review of Agent Oriented software engineering" *International Journal of advanced Research in computer and communication Engineering* Vol.4, issue 4.

Adami, D., Giordana, S., Repeti, M., Coppola, M., Laforenza, D. and Tonellotto, N. (2006). Design and Implementation of a grid network- aware resource broker. In: *proceeding of the IASTED international Conference on parallel and distributed computing and networks as part of the 24th IASTED International Multi-conferences on Applied information.*

Banerjee, A et.al, (2004) "*A time –path scheduling problem (TPSP) for aggregating large data file from Distributed database using an optical Burst-Switched Network*", Proc. ICC 2004, parts, france.

Banerjee, A., Ferg, W., Ghosal, D. and Mukherejee, B. (2008). Algorithms for integrated Routing and Scheduling for aggregating data from distributed resources on a Lambda grid.*IEEE Transaction on parallel and Distributed Systems*.(19) pp.14-34

Banerjee, A., Wu-Chon, F. Senior member IEEE., Dipak, G., Biswanath, M., (2008) Algorithm for integrated Routing and Scheduling for Aggregating data from Distributed Resources on a Lambda Grid. *IEEE Transactions on Parallel and Distributed Systems 19(1)*

Bart, S., Levesque, HJ. And Mitchell, DG. (1992). A New Method for Solving hard. Satisfiability Problem. In: *proceedings of the Tenth National Conference on Artificial Interlligence*. (AAA1-92), San Jose, CA 440-446.

Brakmo, L., Peterson, L. (Oct. 1995) "Tcp Vegas: End to End Congestion Avoidance on a Global Internet". *IEEE Journal on selected Area in Communications, Vol.B. no.8,pp.1465-1480,*

Brown, MD. (Nov.19, 2003). Blue Print for the Future of High Performance Networking Introduction. *Communications ACM*. 46(3).

Bunya, R. (1999). *High Performance Cluster Computer: Architecture and Systems*. Vol.1 and 2. PTR upper saddle River, NJ, USA: Prentice Hall

Buyya, R., David, A. and Jonathan, G., (April10, 2002). Grid resource management Scheduling and Computational Economy. In: *School of Computer Science and Software Engineering, Monash University, Australia*.

Castillo, C., Rouskas, G.N and Hanfoush, K. (2007). On the design of online Scheduling Algorithm for Advance Reservation and Qos in Grids. In: *Processing of 21st IEEE Termination Parallel and Distribution Processing Symposium*, Aveiro, Portugal.

Coffman, E.G., Garey, M., Johnson, D and Lapaugh, S. (Aug.,1985) Scheduling File Transfers. *SIAM Journal on Computing*. 14(3)

Defanti, T., LOAT, C., Mambretti, J., Naggers, K., Arnomd, B., and Ranslight, T. (2003) Global scalar Lambda grid for e-science. In: *Communication of Association of computing machinery*. (46)

DOE: *Ultra Science Net Testbed*. http://www.csm.ornl.gov/ultanet/overview.pdf.

Doug,H.(2003).*National Lambda Rail Inc*. Available at: www.internet2.edu/news/detail/3695

Elizabeth, P., (2003). *Genomes to life Refiners New life from Networks*, US Dept. of Energy (DOE) Workshop. Available at http://www.csm.ornl.gov/ghpn/genome-wk. April14, 2016.

Floyd, S. and Nageswara,.S.V. (Nov.,2008) Dedicated Channels as an Optimal Network Support for Effective Transfer of Massive Data, In: *Proceeding of the 4th ACM/IEEE Symposium on Architectures for Networking and communication system*, San José, California.

Garey, M.R. and David, S.J. (1979). *Computers and Intractability*: *A Grid to the Theory of NP – Completeness.2nd ed.* USA: W.H Freeman

Garey, MR and Johnson, D. (1990). Computers and intractability: *A Grid to the theory of NP-completions*.2nd ed., USA: W.A Freeman.

Green, A., Srikant, R., Whitt, W., (Feb. 1999) "Resource sharing for book-ahead and instantaneous request calls" *IEEE/ACM Transaction on networking,* Vol.7, no.1,pp.10-22.

Gu, Y., Grossman RL., (2004). UDT: UDP- based data transfer for high speed area Network. *Journal of Grid Computing* 1(4), 377-386, In: *proceeding of the 2004 ACM/IEEE Conference on Super Computing.*

Guerin, R. A. and Orda, A. (2000). Network with Advance Reservations: The Routing Perspective In: *Proceeding of Information Communication.* Tel Aviv, Israel.

Hall, N.G and Sriskandarajah, C. (June,1996). A Survey of Machine Scheduling Problems with blocking and no – wait in process. *Journal operations Research* 44 (3), pp. 510-525.

He, E., Leigh, J., Yu, O. and Defanti T.A. (2002). Reliable Blast UDP: *Predicable high performance bulk data transfer.* IEEE 1st Conference on Cluster Computing.

Ho, P-H., and Mouftah, H.T. (2003). A Novel Distributed Protocol for Path Selection in Dynamic Wavelength-Route WDM Network. *Kluwer Photonic Network Communication*. (1): pp 23-32.

Jim, C., Wei, D., and Low, S., (March,2004) "FAST TCP": *Motor ton, architecture, algorithm, performance,*: In proc. of IEEE infocom, Hong-kong.

Kate, C. (2007). Grid Computing and the tera-Grid Gi Science Gateway: *22S-295 High Performance Computing Seminar.*

Lakshmiraman, V. and Ramamurity, B. (2009). Joint Computing and Network Resource Scheduling in a Lambda Grid Network. In: *Proceedings of IEEE International Conference on Communications (ICC 2009).*

Larry, S. (2004). *Optiputer*: Universty of Califonia, San Diego, Ismarr@ucsd.edu. Availble at http://www.evl.uic.edu/cavern/optiputer.

Lee, H., Veeraraghavan, M., Li, H. and E.K. P. Chong (2004). *Lambda Scheduling Algorithm for file transfers on high speed optical circuits*. Available at http://www.ece.virginia.edu/mv/pdf-files/ccgrid-gan04.pdf

Liu, X., Wei, W., Qiao, C., Wang, T., W.Hu, Guo, W., and Hu, W. (2008). Task Scheduling and Light path Establishment in 0ptical Grid. In: *proceeding of the INFOCOM 2008 Mini. Conference Program.* Phoenix, Arizona, USA.

Nagaswara, S.V. Rao, William, R. W, Steven, M. C, and Qishi, W. (2005). Ultra Science Net: *Network Testbed for large-Scale Scince Application*. DOI:10.1109/MCMOM.2005.1541694. Source: IEEE Xplore.

Nagaswara, S.V.Rao, William, G, Bang, Y and Redhakrishnan, S. (2004). *Algorithm for Quickest Path under Different Routing modes*. IEICE Transaction on Communication. 87(4).

Nageswara, .S.V., Wing, W.R., Carter, S.M and Wu, O. (2005). Ultra Science Net: Network Testbed for large – Scale Science Applications. In: *IEEE Communication Magazine*. 43(4)

Nageswara, S.V., Grimmell, W. C., Bang, Y. and Radhakrishnan, S (2004). On Algorithm for Quicka paths under different Routing Modes. In: *IEICE Transaction on Communication,* E87-B 4(3) pp 1002-1006.

Olaf Sporns. (2010) "*Graph theory methods for the Analysis of neural connectivity patterns*" Idiana University, Bloomington, IN47405.

Page, A.J., and Naughton, T.J. (2005). Dynamic task Scheduling using Genetic Algorithm for Heterogeneous Distributed Computing: *Artificial Intelligence Review*. (24).

Pieter T. S, Marc, D. L., Bruno V, Filip, D. T, Bart D, Piet, D. (2007). *Scalable dimensioning of resilient Lambda Grids Science one.* Available at www.sciencedirect.com. April 12, 2015

Pieter, T., Marc, D.L., Bruno, V., Filip, D.T., Bart, Dhoedt., and Piet, D. (2007). *Scalable dimensioning of resilient Lambda Grid.* Available at http://www.dx.doi.org/10.1016/j.future.2007.08.003.

Pinedo, M. L. (2008) Scheduling: *Theory Algorithm and Systems.* 2nd ed. USA: Prentice Hall.

Roger S.(2001) Pressman "Software Engineering Practitioners Approach" Mic Graw hill, fifth edition.

Savera T. W., Lina, B., Harry, P. and Gigi, K. E. (2008*).* Dynamic scheduling of Network Resources with Advance Reservation in Optical Grids. In: *Department of Computer Science Ninth Carolina State University NC, UBA*

She, O., Huang, X., Kannasoot, N., and Zhang, O. (2007). Multi-resource many cost over optical burst switched network. In: *Proceedings of 16th National Conference on Computer Communications and Networks* (ICCCN 2007), Honolulu, Hawaii, USA.

Shen, S., Wei, G., Weiqiang, S., Yaohu, J., Weisheng, H., (Nov.19, 2008) Scheduling and Routing Algorithm for Aggregating Large Data Files from Distributed Databases to Super Computers on Lambda Grid. In: *Proceeding on Network Architectures, management and Applications* V1, 71372D.

Smarr, L., Chien, A., Defanti, T., Leigh, J., Philip, M., and Papadopoulos. (Nov.2003). *The Optiputer Communication of the Association for Computing Machinery*. 46(11), 58-67.

Taesombut, N., Wu, XR., Chien, A., Nayak, A. Smith, B., D.kilb, T.lm, Kent, D., kent, G., and Orcutt, J. (2006). *Collaborative data visualization for Earth Science with the optiputer future Generations Computer system*. 255-63.

Thomas, H. C., Charles, E. Leiserson, Ronald L. Rivest and Clifford, S. (2014). *Introduction to Algorithm*, 2ed. USA: MIT Press.

Travostino, F., Mambretti, J., and Karmons, G.(2006). Grid Network: *Enabling grids with Advanced Communication Technology*. Books.google.com

Veerarghavan, M., Xuau, Z. and Wu, C. (2003) Scheduling and Transport for File Transfers on high-Speed Optical Circuits. *Journal of Grid Computing*. 1 (4)

Veerarghavan, M., Lee, H., Chong, E.K.P., and Li, H. (July, 2004) "*A varying-Bandwidth List Scheduling heuristic for file transfer*" Researchgate DOT: 10.11081cc.

Wang Xia-hong. (2012). *Application of Cisco Packet Tracer Simulation software in Network project Teaching*. pp.32-25. en.cnki.com.cn/Article en/CJFDTOTAL-DNKF201205024.htm.

Wisuik, D. and Greenberg, A., (March, 29-April 2, 1998)" Admission control for booking ahead shared resource" . *In proceedings of IEEE Infocom, Sam Francisco, CA.*

Wolfgang, S. and Behrend, D. (2007). The International VLB1 Service for Geodesy and Astrometry (IVS): Current Capabilities and Future prospects. *Journal of Geodesy.*

WWW.movable-type.co.uk/scripts/latlong.html

APPENDIX A

Scheduler Pseudo code

1: Set of transmission tasks $\{f_1, f_2, \ldots \ldots f_k, \ldots \ldots\}$

2: Set of link resources $L = \{l_1, l_2, \ldots \ldots l_k, \ldots \ldots\}$

3: $l_k's$ **last time interval** $[t_{l_k}, \infty], f_k$'s **transmission time is** t_{f_k}

4: Step1:

5: Sort tasks $f \in F$ into list *LIST*, according to specified priority scheme.

6: **Step2:**

7: ***do***

8: Sort links $l \in L$ into *LIST* , according to scheme that l_k with earlier t_{l_k} is prior

9: ***for all*** $l \in L$ in *LINK* 's priority order

10: ***for all*** $f \in F$ in *LINK* 's priority order

11: ***if*** *f's* path exists at t_l (by running algorithm 2)

12: schedule f to begin transmission at t_l

13: remove f from *LIST*

14: ***for all*** $l_R \in R$

15: $t_{l_R} = t_l + t_f$

16: ***endfor***

17: ***break***

18: ***endif***

19: ***endfor***

20: ***endfor***

21: ***loop while*** there exit tasks in *LIST*

APPENDIX B

Pseudo code for the Path Determination Algorithm.

1: Function FINDPATH (REQUEST R, TOPOLOGY T)

2: i = 1;

3: start time = $T_{(m)}^{i}$

4: end time = $T_{(m)}^{i}$ **+ d**

5: //call transfer time estimate to estimate transfer time using equation (3.1) and (3.12) to find T_f = d (file transfer duration)

6: //and finish time, e respectively.

7: e ⟵ call Transfer Time Estimate (T_f, C, P_d)

8: ***While(end time ≤ e) do***

9: ***While(i ≤ k) do***

10: *find shortest path using* ***Dijkstra's Algorithm***

11: ***if*** *a path is found* ***then***

12: ***if*** *lambda is available on all links during start time and end time*

Then

13: *run **algorithm 2** to locate lambdas*

14: ***return***

*15**:*** ***else***

16: *delete busy link from topology*

*17**:*** ***i***++

18: **endif**

19: **endif**

20: ***end While***

21: Start time = start time + t

22: Start time = start time + d

23**:** **end While**

24: **end function**

APPENDIX C

Lambda Grid Scheduler Source Code

shortest_path.h:

lambda_schedule.h:

File_aggregation_calculat.h:

```
#ifndef LAMBDA_GRID_ROUTING_SHORTEST_PATH_H
#define SHORTEST_PATH_H

#include <queue>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <vector>

namespace lambdaGridNetworkScheduler {
    template<class NodeType>
    class TransmissionTaskAnalyzer {
    protected:
```

```
    using Set = std::unordered_set<NodeType*>;

  public:

    float job_Blocking_Rate;
    float Tmin; #minimum file aggregation time from supercomputer
    String Target;
    Float Route_Scheduler_metric;
    Float Total_File_Transfer_Time;

     TransmissionTaskAnalyzer(std::string name) : m_name{name} {}
     virtual void CreatFileTransmissionFileList(NodeType* other) = 0;
     virtual bool LambdaGrid_is_connected_to(NodeType* other) const = 0;
     virtual void LambdaGrid_disconnect_from(NodeType* other) = 0;
     virtual typename Set::iterator begin() const = 0;
     virtual typename Set::iterator end() const = 0;

#Create iterator for File transfer Aggregation # File transfer begin time && file transfer
        end time
```

```
class ParentIterator {
public:

    ParentIterator() : mp_set{nullptr} {}

    typename Set::iterator begin()
    {
        return mp_set->begin();
    }
    typename Set::iterator end()
    {
        return mp_set->end();
    }
    void set_list(Set* p_list)
    {
        this->mp_set = p_list;
    }

private:

    std::unordered_set<NodeType*>* mp_set;
```

```
    };

    virtual ParentIterator* parents() = 0;

    bool operator==(const NodeType& other) const
    {
        return m_name == other.m_name;
    }

    std::string& get_name() {return m_name;}

protected:

    std::string m_name;
};

template<class T, class FloatType = double>
class AbstractLambdaGridWeightFunction {
public:

    virtual FloatType& operator()(T* Lambda_node1, T* Lamda_node2) = 0;
```

```
};


template<class FloatType>
class File_Aggregation_Vector {
private:
    const FloatType m_x;
    const FloatType m_y;
    const FloatType m_z;

public:
    Point3D(const FloatType x = FloatType(),
            const FloatType y = FloatType(),
            const FloatType z = FloatType())
                        :
                        m_x{x},
                        m_y{y},
                        m_z{z} {}

    FloatType x() const {return m_x;}
    FloatType y() const {return m_y;}
    FloatType z() const {return m_z;}
```

```
#Initialize link access with Supercomputer

int Tf, sF, Pdf. Oc;

 };

 template<class FloatType>
 class Routing_AbstractMetric {
 public:

    virtual FloatType operator()(coderodde::Point3D<FloatType>& p1,
                                 coderodde::Point3D<FloatType>& p2) = 0;
 };

 template<class FloatType>
 class EuclideanMetric : public coderodde::AbstractMetric<FloatType> {
 public:

    FloatType operator()(coderodde::Point3D<FloatType>& p1,
                         coderodde::Point3D<FloatType>& p2) {
        const FloatType dx = p1.x() - p2.x();
```

```
        const FloatType dy = p1.y() - p2.y();
        const FloatType dz = p1.z() - p2.z();

        return std::sqrt(dx * dx + dy * dy + dz * dz);
    }
};

template<class T, class FloatType = double>
class LayoutMap {
public:

    virtual coderodde::Point3D<FloatType>*& operator()(T* key)
    {
        return m_map[key];
    }

    ~LayoutMap()
    {
        typedef typename std::unordered_map<T*,
                        coderodde::Point3D<FloatType>*>::iterator it_type;
        for (it_type iterator = m_map.begin();
```

```
            iterator != m_map.end(); iterator++)
        {
            delete iterator->second;
        }
    }

  private:

    std::unordered_map<T*, coderodde::Point3D<FloatType>*> m_map;
};

/* File transfer conversion computation */
int file_Transfer_socket (FILE *fp, int sockfd, long fsize)
{
        LARGE_INTEGER freq, start, end;
        double speed, rate, eta;
        char buf[16*1024];
        long tot;
        int i, n;

        /* Schedule & Frequency */
```

```
if (!QueryPerformanceFrequency(&freq))
        return -1;

/* Start time */
if (!QueryPerformanceCounter(&start))
        return -1;

/* Main loop */
for (tot = 0; tot < filesize;)
{
        if ((n = fread(buf, 1, sizeof(buf), fp)) <= 0)
        {
                if (feof(fp))
                        break;
                else
                        return -1;
        }

        /* Sendall */
        if ((i = sendall(sockfd, buf, n)) != n)
                return -1;
```

```
        /* End time */
        if (!QueryPerformanceCounter(&end))
            return -1;

        /* Calculate */
        tot += i;
        speed = (double)(end.QuadPart - start.QuadPart) / (double)freq.QuadPart;
        rate = (double)(tot / speed) / 1024; // KiB/s
        eta = (double)(fsize - tot) / (double)rate / 60; // Minutes

        printf("log: Sent: %ld of %ld Prog: %.2f%% Rate: %.1fKiB/s ETA:
    %.1f\r",
            tot, fsize, ((double)tot *100.0) / (double)fsize, rate, eta);

        fflush(stdout);
    }

    return tot;
}
```

```
template<class NodeType, class DistanceType = double>
class LambdaGrid_Network_State_Analyzer{
public:
   LambdaGrid_Network_State_Analyzer(NodeType*          Supercomputer_node,
      LinkCapacity distance) :
       mp_node{p_node},
       m_route_distance{distance} {}

   NodeType* get_node()
   {
      return FileTransferLink_node;
   }

   DistanceType get_FileTransferLink_node()
   {
      return m_FileTransferLink_node;
   }

private:
```

```
        NodeType*
distination_node;
        DistanceType distination_distance;
    };

    template<class NodeType, class DistanceType = double>
    class TPSP_Route_Computation {
    public:

        bool operator()(HeapNode<NodeType, LambdaGrid_DistanceType>* p_first,
                        LambdaGrid_DistanceTyp<NodeType, DistanceType>* p_second)
        {
            return p_first->get_distance() > p_second->get_distance();
        }
    };

    template<class NodeType, class FloatType = double>
    class TaskCoordinator {
    public:

        FloatType& operator()(const NodeType* p_node)
```

```
    {
        return m_map[p_node];
    }

private:
 Tf=

    std::unordered_map<const NodeType*, FloatType> m_map;
};

template<class NodeType>
class ParentMap {
public:

    NodeType*& operator()(const NodeType* p_node)
    {
        return m_map[p_node];
    }

    bool has(NodeType* p_node)
    {
```

```
        return m_map.find(p_node) != m_map.end();
    }

private:

    std::unordered_map<const NodeType*, NodeType*> m_map;
};

template<class NodeType>
std::vector<NodeType*>* traceback_path(NodeType* p_touch,
                                       ParentMap<NodeType>* parent_map1,
                                       ParentMap<NodeType>* parent_map2 = nullptr)
{
    std::vector<NodeType*>* p_path = new std::vector<NodeType*>();
    NodeType* p_current = p_touch;

    while (p_current != nullptr)
    {
        p_path->push_back(p_current);
        p_current = (*parent_map1)(p_current);
    }
```

```
    std::reverse(p_path->begin(), p_path->end());

    if (parent_map2 != nullptr)
    {
        p_current = (*parent_map2)(p_touch);

        while (p_current != nullptr)
        {
            p_path->push_back(p_current);
            p_current = (*parent_map2)(p_current);
        }
    }

    return p_path;
}

template<class T, class FloatType = double>
class HeuristicFunction {

public:
```

```
    HeuristicFunction(T* p_target_element,
                      LayoutMap<T, FloatType>& layout_map,
                      AbstractMetric<FloatType>& metric)
    :
    mp_layout_map{&layout_map},
    mp_metric{&metric},
    mp_target_point{layout_map(p_target_element)}
    {


    }


    FloatType operator()(T* element)
    {
        return (*mp_metric)(*(*mp_layout_map)(element), *mp_target_point);
    }


private:
    coderodde::LayoutMap<T, FloatType>*   mp_layout_map;
    coderodde::AbstractMetric<FloatType>* mp_metric;
    coderodde::Point3D<FloatType>*        mp_target_point;
};
```

```
template<class NodeType, class WeightType = double>
std::vector<NodeType*>*
astar(NodeType* p_source,
      NodeType* p_target,
      coderodde::AbstractWeightFunction<NodeType, WeightType>& w,
      coderodde::LayoutMap<NodeType, WeightType>& layout_map,
      coderodde::AbstractMetric<WeightType>& metric)
{
    std::priority_queue<HeapNode<NodeType, WeightType>*,
                        std::vector<HeapNode<NodeType, WeightType>*>,
                        HeapNodeComparison<NodeType, WeightType>> OPEN;

    std::unordered_set<NodeType*> CLOSED;

    coderodde::HeuristicFunction<NodeType,
                                 WeightType> h(p_target,
                                               layout_map,
                                               metric);
    DistanceMap<NodeType, WeightType> d;
    ParentMap<NodeType> p;
```

```
OPEN.push(new LambdaHeapNode<NodeType, WeightType>(p_source,
  WeightType(0)));
p(p_source) = nullptr;
d(p_source) = WeightType(0);
 String file_schedule ;

while (!OPEN.empty())
{
   LambdaHeapNode<NodeType, WeightType>* p_heap_node = OPEN.top();
   NodeType* p_current = Lambda_p_heap_node->get_Supercomputer_node();
   OPEN.pop();
   delete p_heap_node;

   if (*p_current == *p_target)
   {
      // Found the path.
      return traceback_path(p_target, &p);
   }

   CLOSED.insert(p_current);
```

```
// For each child of 'p_current' do...
for (NodeType* p_child_Supercomputer : *p_current)
{

    if (CLOSED.find(p_child) != CLOSED.end())
    {
        // The optimal distance from source to p_child is known.
        continue;
    }

    WeightType cost = d(p_current) + w(p_current, p_child);

    if (!p.has(p_child) || cost < d(p_child))
    {
        WeightType f = cost + h(p_child);
        OPEN.push(new                    HeapNode_Supercomputer<NodeType,
WeightType>(p_child, f));
        d(p_child) = cost_Supercomputer;
        p(p_child) = p_current;
    }
```

```
        }
    }

    // p_target not reachable from p_source_Supercomputer.
    return nullptr;
}
template<class T, class FloatType>
class ConstantLayoutMap_Supercomputer_File_Transfer_Schedule : public
    coderodde::LayoutMap<T, FloatType> {
public:

    ConstantLayoutMap() : mp_point{new Point3D<FloatType>()} {}

    ~ConstantLayoutMap()
    {
        delete mp_point;
    }

    Point3D<FloatType>*& operator()(T* key)
    {
        return mp_point;
```

```
    }

private:

    Point3D<FloatType>* mp_point;
};

    /*****************************************************************
    *****************************************************************
* This function template implements Dijkstra's shortest path algorithm. For the
    Lambda Grid Schedulling    *

    *****************************************************************
    *****************************************************************
    /
template<class NodeType, class WeightType = double>
std::vector<NodeType*>*
dijkstra(NodeType* File_Transfer_source,
        NodeType* File_Transfer_target,
        coderodde::AbstractWeightFunction<NodeType, WeightType>& w)
{
```

```
    ConstantLayoutMap_Supercomputer_File_Transfer_Schedule<NodeType,
      WeightType> layout;
    EuclideanMetric_Supercomputer_File_Transfer_Schedule<WeightType> metric;


    return astar(p_source,
                 p_target,
                 w,
                 layout,
                 metric);
}

template<class NodeType, class WeightType = double>
std::vector<NodeType*>*
bidirectional_dijkstra(
    NodeType* p_source,
    NodeType* p_target,

      coderodde::AbstractWeightFunction_Supercomputer_File_Transfer_Schedule<No
      deType, WeightType>& w)
{
    std::priority_queue<Lambda_HeapNode<NodeType, WeightType>*,
```

```
            std::vector<HeapNode<NodeType, WeightType>*>,
            HeapNodeComparison<NodeType, WeightType>> OPENA;

std::priority_queue<ambda_HeapNode<NodeType, WeightType>*,
            std::vector<HeapNode<NodeType, WeightType>*>,
            HeapNodeComparison<NodeType, WeightType>> OPENB;

std::unordered_set<NodeType*> CLOSEDA;
std::unordered_set<NodeType*> CLOSEDB;

DistanceMap<NodeType, WeightType> DISTANCEA;
DistanceMap<NodeType, WeightType> DISTANCEB;

ParentMap<NodeType> PARENTA;
ParentMap<NodeType> PARENTB;

OPENA.push(new HeapNode<NodeType, WeightType>(p_source, 0.0));
OPENB.push(new HeapNode<NodeType, WeightType>(p_target, 0.0));

DISTANCEA(p_source) = WeightType(0);
DISTANCEB(p_target) = WeightType(0);
```

```
PARENTA(p_source) = nullptr;
PARENTB(p_target) = nullptr;

NodeType* p_touch = nullptr;
WeightType best_cost = std::numeric_limits<WeightType>::max();

while (!OPENA.empty() && !OPENB.empty())
{
   if (OPENA.top()->get_distance() +
      OPENB.top()->get_distance() >= best_cost)
   {
      return traceback_path(p_touch, &PARENTA, &PARENTB);
   }

   if (OPENA.top()->get_distance() < OPENB.top()->get_distance())
   {
      Lambda_HeapNode<NodeType, WeightType>* p_heap_node = OPENA.top();
      NodeType* p_current = p_heap_node->get_node();
      OPENA.pop();
      delete p_heap_node;
```

```
CLOSEDA.insert(p_current);

for (NodeType* p_child : *p_current)
{
   if (CLOSEDA.find(p_child) != CLOSEDA.end())
   {
      continue;
   }

   WeightType g = DISTANCEA(p_current) + w(p_current, p_child);

   if (!PARENTA.has(p_child) || g < DISTANCEA(p_current))
   {
      OPENA.push(new HeapNode<NodeType,
                              WeightType>(p_child, g));
      DISTANCEA(p_child) = g;
      PARENTA(p_child) = p_current;

      if (CLOSEDB.find(p_child) != CLOSEDB.end())
      {
```

```
            WeightType path_len = g + DISTANCEB(p_child);

            if (best_cost > path_len)
            {
               best_cost = path_len;
               p_touch = p_child;
            }
         }
      }
   }
}
else
{
   Lambda_HeapNode<NodeType, WeightType>* p_heap_node = OPENB.top();
   NodeType* p_current = p_heap_node->get_node();
   OPENB.pop();
   delete p_heap_node;

   CLOSEDB.insert(p_current);

   typename coderodde::AbstractGraphNode<NodeType>::ParentIterator*
```

```
        p_iterator = p_current->parents();

for (NodeType* p_parent : *p_iterator)
{
    if (CLOSEDB.find(p_parent) != CLOSEDB.end())
    {
        continue;
    }

    WeightType g = DISTANCEB(p_current) +
                w(p_parent, p_current);

    if (!PARENTB.has(p_parent) || g < DISTANCEB(p_parent))
    {
        OPENB.push(new HeapNode<NodeType,
                            WeightType>(p_parent, g));
        DISTANCEB(p_parent) = g;
        PARENTB(p_parent) = p_current;

        if (CLOSEDA.find(p_parent) != CLOSEDA.end())
        {
```

```
                    WeightType path_len = g + DISTANCEA(p_parent);


                    if (best_cost > path_len)
                    {
                        best_cost = path_len;
                        p_touch = p_parent;
                    }
                }
            }
        }
    }

    return nullptr;
}

class DirectedGraphNode : public
    coderodde::AbstractGraphNode_Supercomputer_File_Transfer_Schedule<Directe
    dGraphNode> {
public:
```

```
DirectedGraphNode_Supercomputer_File_Transfer_Schedule(std::string name) :
      coderodde::AbstractGraphNode<DirectedGraphNode>(name)
{
   this->m_name = name;
}

void connect_to(coderodde::DirectedGraphNode* p_other)
{
   m_out.insert(p_other);
   p_other->m_in.insert(this);
}

bool is_connected_to(coderodde::DirectedGraphNode* p_other) const
{
   return m_out.find(p_other) != m_out.end();
}

void disconnect_from(coderodde::DirectedGraphNode* p_other)
{
   m_out.erase(p_other);
   p_other->m_in.erase(this);
```

```
}

ParentIterator* parents()
{
   m_iterator.set_list(&m_in);
   return &m_iterator;
}

typename Set::iterator begin() const
{
   return m_out.begin();
}

typename Set::iterator end() const
{
   return m_out.end();
}

friend std::ostream& operator<<(std::ostream& out,
                                DirectedGraphNode& node)
{
```

```
        return out << "[DirectedGraphNode " << node.get_name() << "]";
    }

private:
    Set m_in;
    Set m_out;
    ParentIterator m_iterator;
};

class DirectedGraphWeightFunction :
public
        AbstractWeightFunction_Supercomputer_File_Transfer_Schedule<coderodde::Dir
        ectedGraphNode, double> {

public:

    double& operator()(coderodde::DirectedGraphNode* node1,
                       coderodde::DirectedGraphNode* node2)
    {
        if (m_map.find(node1) == m_map.end())
        {
```

```
            m_map[node1] =
            new std::unordered_map<coderodde::DirectedGraphNode*,
                                   double>();
        }

        return (*m_map.at(node1))[node2];
    }

  private:

    std::unordered_map<coderodde::DirectedGraphNode*,
    std::unordered_map<coderodde::DirectedGraphNode*, double>*> m_map;
  };
}

#endif // LAMBDA_GRID_FILE_SHORTEST_PATH_H
```

main.cpp:

```
#include <iostream>
#include <random>
#include <string>
```

```
#include <tuple>
#include <vector>

#include "File_Aggregation_shortest_path.h"

using std::cout;
using std::endl;
using std::get;
using std::make_tuple;
using std::mt19937;
using std::random_device;
using std::string;
using std::to_string;
using std::tuple;
using std::vector;
using std::uniform_int_distribution;
using std::uniform_real_distribution;

using std::chrono::duration_cast;
using std::chrono::milliseconds;
using std::chrono::system_clock;
```

```
using coderodde::astar;
using coderodde::bidirectional_dijkstra;
using coderodde::dijkstra;
using coderodde::DirectedGraphNode;
using coderodde::DirectedGraphWeightFunction;
using coderodde::EuclideanMetric;
using coderodde::HeuristicFunction;
using coderodde::LayoutMap;
using coderodde::Point3D;

/*************************************************************************
        ********
* Randomly selects an element from a lambda grid vector.                 *
*************************************************************************
        *******/
template<class T>
T& choose(vector<T>& vec, mt19937& rnd_gen)
{
    uniform_int_distribution<size_t> dist(0, vec.size() - 1);
    return vec[dist(rnd_gen)];
```

```
}

/*****************************************************************************
        ********
* Creates a random point in a plane.                                         *
*****************************************************************************
        *******/
static Point3D<double>* create_random_point(const double xlen,
                                            const double ylen,
                                            mt19937& random_engine)
{
   uniform_real_distribution<double> xdist(0.0, xlen);
   uniform_real_distribution<double> ydist(0.0, ylen);

   return new Point3D<double>(xdist(random_engine),
                              ydist(random_engine),
                              0.0);
}

/*****************************************************************************
        ********
```

```
* Creates a random directed, weighted graph.                          *
************************************************************************
        *******/
static tuple<vector<DirectedGraphNode*>*,
             DirectedGraphWeightFunction*,
             LayoutMap<DirectedGraphNode, double>*>
   create_random_graph(const size_t length,
                       const double area_width,
                       const double area_height,
                       const float arc_load_factor,
                       const float distance_weight,
                       mt19937 random_gen)
{
   vector_Supercomputer_File_Transfer_Schedule<DirectedGraphNode*>*  p_vector  =
        new vector<DirectedGraphNode*>();
   LayoutMap<DirectedGraphNode, double>* p_layout =
   new          LayoutMap_Supercomputer_File_Transfer_Schedule<DirectedGraphNode,
        double>();

   for (size_t i = 0; i < length; ++i)
   {
```

```
   DirectedGraphNode*                p_node                =                new
      Lambda_Grid_DirectedGraphNode(to_string(i));
   p_vector->push_back(p_node);
}

for (Lambda_Grid_DirectedGraphNode* p_node : *p_vector)
{
   Point3D<double>* p_point = create_random_point(area_width,
                                    area_height,
                                    random_gen);
   (*p_layout)(p_node) = p_point;
}

_Supercomputer_File_Transfer_ScheduleDirectedGraphWeightFunction* p_wf = new
      DirectedGraphWeightFunction();
EuclideanMetric<double> euclidean_metric;

size_t arcs = arc_load_factor > 0.9 ?
length * (length - 1) :
(arc_load_factor < 0.0 ? 0 : size_t(arc_load_factor * length * length));
```

```
  while (arcs > 0)
  {
    Lambda_Grid_DirectedGraphNode* p_head = choose(*p_vector, random_gen);
    Lambda_Grid_DirectedGraphNode* p_tail = choose(*p_vector, random_gen);

    Point3D<double>* p_head_point = (*p_layout)(p_head);
    Point3D<double>* p_tail_point = (*p_layout)(p_tail);

    const double cost = euclidean_metric(*p_head_point,
                                         *p_tail_point);


    (*p_wf)(p_tail, p_head) = distance_weight * cost;
    p_tail->connect_to(p_head);

    --arcs;
  }

  return make_tuple(p_vector, p_wf, p_layout);
}
```

```
/*****************************************************************************
        ********
* Returns the amount of milliseconds since file transfer initiated                *
*****************************************************************************
        *******/
static unsigned long long get_milliseconds()
{
   return duration_cast<milliseconds>(system_clock::now()
                                .time_since_epoch()).count();
}

/*****************************************************************************
        ********
* Checks that a lambda grid path has all needed arcs.                             *
*****************************************************************************
        *******/
static bool is_valid_path(vector<DirectedGraphNode*>* p_path)
{
   for (size_t i = 0; i < p_path->size() - 1; ++i)
   {
      if (!(*p_path)[i]->is_connected_to((*p_path)[i + 1]))
```

```
        {
            return false;
        }
    }

    return true;
}

/*************************************************************************
        ********
* Computes the length (cost) of a path.                                  *
*************************************************************************
        *******/
static double compute_path_length(vector<DirectedGraphNode*>* p_path,
                                  DirectedGraphWeightFunction* p_wf)
{
    double cost = 0.0;

    for (size_t i = 0; i < p_path->size() - 1; ++i)
    {
        cost += (*p_wf)(p_path->at(i), p_path->at(i + 1));
```

```
    }

    return cost;
}

/**************************************************************************
        ********
                                            *
**************************************************************************
        *******/
int main(int argc, const char * argv[]) {
    random_device rd;
    mt19937 random_gen(rd());

    cout << "Building a graph..." << endl;

    tuple<vector<Lambda_Grid_DirectedGraphNode*>*,
        DirectedGraphWeightFunction*,
        LayoutMap<DirectedGraphNode, double>*> graph_data =
    create_random_graph(50000,
                1000.0,
```

```
                700.0,
                0.0001f,
                1.2f,
                random_gen);

Lambda_Grid_DirectedGraphNode *const p_source =
    choose(*std::get<0>(graph_data),
                random_gen);

Lambda_Grid_DirectedGraphNode *const p_target =
    choose(*std::get<0>(graph_data),
                random_gen);

cout << "Source: " << *Supercomputer_source << endl;
cout << "Target: " << *Supercomputer_target << endl;

EuclideanMetric<double> em;

unsigned long long ta = get_milliseconds();

vector<Lambda_Grid_DirectedGraphNode*>* p_path1 =
```

```
        astar(p_source,
              p_target,
              *get<1>(graph_data),
              *get<2>(graph_data),
              em);

unsigned long long tb = get_milliseconds();

cout << endl;
cout << "A* path:" << endl;

if (!p_path1)
{
   cout << "No path for A*!" << endl;
   return 0;
}

for (Lambda_Grid_DirectedGraphNode* p_node : *p_path1)
{
   cout << *p_node << endl;
}
```

```
cout << "Time elapsed: " << tb - ta << " ms." << endl;

cout << std::boolalpha;

cout << "Is valid path: " << is_valid_path(p_path1) << endl;

cout << "Cost: " << compute_path_length(p_path1, get<1>(graph_data)) << endl;


cout << endl;

cout << "File Arrival rate:" << endl;


ta = get_milliseconds();


vector<Lambda_Grid_DirectedGraphNode*>* p_path2 =

       dijkstra(p_source,

                p_target,

                *get<1>(graph_data));


tb = get_milliseconds();


if (!p_path2)

{

   cout << "No path for Dijkstra's algorithm!" << endl;
```

```
    return 0;
}

for (Lambda_Gride_DirectedGraphNode* p_node : *p_path2)
{
    cout << *p_node << endl;
}

cout << "Time elapsed: " << tb - ta << " ms." << endl;
cout << "Is valid path: " << is_valid_path(p_path2) << endl;
cout << "Cost: " << compute_path_length(p_path2, get<1>(graph_data)) << endl;

cout << endl;
cout << "Bidirectional Dijkstra path:" << endl;

ta = get_milliseconds();

vector<DirectedGraphNode*>* p_path3 =
        bidirectional_dijkstra(p_source,
                               p_target,
                               *get<1>(graph_data));
```

```
tb = get_milliseconds();

if (!p_path3)
{
    cout << "No path for bidirectional Dijkstra's algorithm!" << endl;
    return 0;
}

for (DirectedGraphNode* p_node : *p_path3)
{
    cout << *p_node << endl;
}

cout << "Time elapsed: " << tb - ta << " ms." << endl;
cout << "Is valid path: " << is_valid_path(p_path3) << endl;
cout << "Cost: " << compute_path_length(p_path3, get<1>(graph_data)) << endl;

vector<coderodde::DirectedGraphNode*>* p_vec = get<0>(graph_data);

while (!p_vec->empty())
{
```

```
        delete p_vec->back();
        p_vec->pop_back();
    }

    delete get<0>(graph_data);
    delete get<1>(graph_data);
    delete get<2>(graph_data);

    return 0;
}
```

APPENDIX D1

Simulation processing trace file

ITERATION 1

Building configuration...

Current configuration: 977 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer1

!

!

!

sample interval set 8 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource schedule count 32.4 !!! count wave length 7 !!! Delay 20 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transfered 2.6887 megabytes

CPU time 033.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 1: begin file aggregation 06/4/2016 4:57:32.02 AM end file aggregation 06/4/2016 5:07:36.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 1: begin file aggregation 06/4/2016 9:45:07.04 AM end file aggregation 06/24/2016 9:45:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 4: begin file aggregation 06/24/2016 9:56:13.00 AM end file aggregation 06/24/2016 9:56:16.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 7: begin file aggregation 06/24/2017 10:02:06.01.00 AM end channel sense 06/24/2017 10:02:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D2

Simulation processing trace file

ITERATION 2

Building configuration...

Current configuration: 977 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer2

!

!

!

sample interval set 8 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource schedule count 32.4 !!! count wave length 7 !!! Delay 18 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 11,4,01,13

buffer-size 8bggg00234 bytes

data transfered 3.34487 megabytes

CPU time 033.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 2: begin file aggregation 06/4/2016 5:07:37.02 AM end file aggregation 06/4/2016 5:57:36.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 2: begin file aggregation 06/4/2016 10:45:07.04 AM end file aggregation 06/24/2016 10:47:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 4: begin file aggregation 06/24/2016 10:56:13.00 AM end file aggregation 06/24/2016 10:58:16.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 7: begin file aggregation 06/24/2017 11:02:06.01.00 AM end channel sense 06/24/2017 11:02:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D3

Simulation processing trace file

ITERATION 3

Building configuration...

Current configuration: 977 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer3

!

!

!

sample interval set 9 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource schedule count 32.4 !!! Count wave length 7 !!! Delay 18 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transfered 2.8899 megabytes

CPU time 033.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 1: begin file aggregation 06/4/2016 5:17:32.02 AM end file aggregation 06/4/2016 5:37:36.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 1: begin file aggregation 06/4/2016 10:15:07.04 AM end file aggregation 06/24/2016 10:40:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 4: begin file aggregation 06/24/2016 10:56:13.00 AM end file aggregation 06/24/2016 10:56:16.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 10: begin file aggregation 06/24/2017 11:12:06.01.00 AM end channel sense 06/24/2017 11:22:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D4

Simulation processing trace file

ITERATION 4

Building configuration...

Current configuration: 777 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer4

!

!

!

sample interval set 8 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource scedule count 32.4 !!! count wave length 7 !!! Delay 11 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transfered 6.6988 megabytes

CPU time 033.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 1: begin file aggregation 06/4/2016 5:52:32.02 AM end file aggregation 06/4/2016 6:17:36.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 1: begin file aggregation 06/4/2016 10:15:07.04 AM end file aggregation 06/24/2016 10:45:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 4: begin file aggregation 06/24/2016 10:16:13.00 AM end file aggregation 06/24/2016 10:56:16.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 7: begin file aggregation 06/24/2017 11:02:06.01.00 AM end channel sense 06/24/2017 11:08:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D5

Simulation processing trace file

ITERATION 5

Building configuration...

Current configuration: 917 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer5

!

!

!

sample interval set 8 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource schedule count 32.4 !!! count wave length 7 !!! Delay 13 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transfered 8.6989 megabytes

CPU time 033.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 5: begin file aggregation 06/4/2016 4:57:32.02 AM end file aggregation 06/4/2016 5:07:36.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 5: begin file aggregation 06/4/2016 10:40:07.04 AM end file aggregation 06/24/2016 10:45:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 4: begin file aggregation 06/24/2016 10:46:13.00 AM end file aggregation 06/24/2016 10:56:16.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 7: begin file aggregation 06/24/2017 11:02:19.01.00 AM end channel sense 06/24/2017 11:10:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D6

Simulation processing trace file

ITERATION 6

Building configuration...

Current configuration: 977 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer6

!

!

!

sample interval set 8 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource schedule count 32.4 !!! count wave length 7 !!! Delay 08 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transfered 3.6787 megabytes

CPU time 022.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 6: begin file aggregation 06/4/2016 4:57:32.02 AM end file aggregation 06/4/2016 5:07:36.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 6: begin file aggregation 06/4/2016 10:25:07.04 AM end file aggregation 06/24/2016 10:45:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 6: begin file aggregation 06/24/2016 9:36:13.00 AM end file aggregation 06/24/2016 9:56:16.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 7: begin file aggregation 06/24/2017 10:02:06.01.00 AM end channel sense 06/24/2017 10:12:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D7

Simulation processing trace file

ITERATION 7

Building configuration...

Current configuration: 977 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer7

!

!

!

sample interval set 8 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource schedule count 32.4 !!! count wave length 7 !!! Delay 18 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transferred 10.6993 megabytes

CPU time 013.000025 seconds

date/time format mm/dd/yyyy

Supercomputer 1: begin file aggregation 06/4/2016 5:57:42.02 AM end file aggregation 06/4/2016 6:07:36.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 1: begin file aggregation 06/4/2016 9:45:07.04 AM end file aggregation 06/24/2016 9:45:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 4: begin file aggregation 06/24/2016 10:56:13.00 AM end file aggregation 06/24/2016 10:18:16.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 9: begin file aggregation 06/24/2017 10:02:06.01.00 AM end channel sense 06/24/2017 10:09:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D8

Simulation processing trace file

ITERATION 8

Building configuration...

Current configuration: 977 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer8

!

!

!

sample interval set 8 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource schedule count 32.4 !!! count wave length 7 !!! Delay 16 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transfered 8.9982 megabytes

CPU time 033.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 2: begin file aggregation 06/4/2016 6:27:32.02 AM end file aggregation 06/4/2016 8:07:46.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 1: begin file aggregation 06/4/2016 10:55:17.04 AM end file aggregation 06/24/2016 11:12:10.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 5: begin file aggregation 06/24/2016 11:56:13.00 AM end file aggregation 06/24/2016 12:16:16.13 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 8: begin file aggregation 06/24/2017 10:02:06.01.00 AM end channel sense 06/24/2017 10:10:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D9

Simulation processing trace file

ITERATION 9

Building configuration...

Current configuration: 987 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer9

!

!

!

sample interval set 9 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource schedule count 32.4 !!! count wave length 7 !!! Delay 11 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transfered 7.6982 megabytes

CPU time 033.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 9: begin file aggregation 06/4/2016 4:57:32.02 AM end file aggregation 06/4/2016 5:07:36.12 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 1: begin file aggregation 06/4/2016 6:15:07.04 AM end file aggregation 06/24/2016 7:45:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 10: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 8: begin file aggregation 06/24/2016 11:16:13.00 AM end file aggregation 06/24/2016 12:56:16.17 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 5: begin file aggregation 06/24/2017 10:02:06.01.00 AM end channel sense 06/24/2017 10:02:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!

APPENDIX D10

Simulation processing trace file

ITERATION 10

Building configuration...

Current configuration: 987 bytes

!

version 12.1

no service timestamps log date/time msec

no service timestamps debug date/time msec

no service password-encryption

!

hostname mesh Supercomputer10

!

!

!

sample interval set 8 seconds

optical interface 1EEE FX77703.34 !!!

optical channels set.....channel : 13 fx8976.9864 !!! channel optical channel OC_192 15665000000 .....

data transfer threshold 10GB:

lambda_1_traffic : !!! !!! !!! !!! !!! !!! !!!

resource scedule count 32.4 !!! count wave length 7 !!! Delay 18 wait_queue 67543

lambda Grid channel !!! Supercomputer online Teraflops 34555000033,444559825,845345455,3453456,75345345435,104545,1255555,253453,953453 Target Aggregate 1,4,11,13

buffer-size 8bggg00234 bytes

data transfered 12.6989 megabytes

CPU time 033.000045 seconds

date/time format mm/dd/yyyy

Supercomputer 1: begin file aggregation 06/4/2016 8:17:32.02 AM end file aggregation 06/4/2016 9:17:16.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 6: begin file aggregation 06/4/2016 9:45:07.04 AM end file aggregation 06/24/2016 9:45:10.02 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 4: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 8: optical path traffic mask

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

Supercomputer 4: begin file aggregation 06/24/2016 11:16:23.00 AM end file aggregation 06/24/2016 11:56:16.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

:

Supercomputer 7: begin file aggregation 06/24/2017 11:12:06.11.00 AM end channel sense 06/24/2017 12:02:9.33.03 AM

File aggregation switch!!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!! !!!

total system roundtriptime

!

!

!

!

!

!

!

!

!

!

!

!