2010

# Data transfer scheduling with advance reservation and provisioning

Mehmet Balman
*Louisiana State University and Agricultural and Mechanical College*

# DATA TRANSFER SCHEDULING WITH ADVANCE RESERVATION AND PROVISIONING

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Mehmet Balman
B.S., Bogazici University, 2000
M.S., Bogazici University, 2006
M.S., Louisiana State University, 2008

August 2010

# Acknowledgments

As a doctoral student, I had been privileged to work with my committee chair, Tevfik Kosar, during my study at Louisiana State University. As his first student, I benefited from his instructive comments and his mentorship in academic and scientific approach, that helped me expand my research horizon. I have acquired great knowledge and research skills by working with him. I would also like to mention my great appreciation for my committee members, Gabrielle Allen, Konstantin Busch, Jianhua Chen and Ramachandran Vaidyanathan. They have supported and encouraged me in many ways.

I am heartily thankful to Evangelos Triantaphyllou for his extremely valuable guidance. I have always admired his high quality of scholarship, and I appreciate his effort as a friendly student advisor, and his advice helped me a lot.

I would like to thank Thomas Sterling for his thoughtful comments during our discussions in the HPC area in general, which helped me better focus specifically on my topic. I would also like to mention Daniel S. Katz for his scholarly comments and help in the early phases of my research.

I would like to thank Hugh Greenberg and Lachesar Ionkov from Los Alamos National Laboratory. I had the great opportunity to meet with David Daniel and Adolfy Hoisie at LANL. I would also like to express my great appreciation for John Bent, James Nunez, and Gary Grider for their valuable comments and support. It was a great pleasure for me to have met them, and to have discussed future directions that encouraged me to shape my work.

I owe my deepest gratitude to my senior colleagues at Lawrence Berkeley National Laboratory for their encouragement and guidance, that helped me complete this work. It is an honor for me to continue working with Arie Shoshani. I have been impressed with his ability to structure the fundamental concepts and see beyond, and to target future research problems. I also express my deepest appreciation for Alex Sim. He has provided encouragement and support, and guided me in many ways since I started working with him last summer. This dissertation would not have been possible without the support and help from both of them.

I thank my parents, Sahap Balman and Nursel Balman, who patiently supported me in all respect

throughout my entire study. I am also very much grateful to my dearest friends within and outside the CS department, who made me enjoy my graduate life. Especially, Promita Chakraborty has always been ready to help and support me in every stage of my work.

This work is in part supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract no. DE-AC02-05CH11231, and in part supported by the National Science Foundation under award numbers CNS-0619843, CNS-0846052, OCI-0926701 and EPS-0701491.

# Table of Contents

# Abstract

Over the years, scientific applications have become more complex and more data intensive. Although through the use of distributed resources the institutions and organizations gain access to the resources needed for their large-scale applications, complex middleware is required to orchestrate the use of these storage and network resources between collaborating parties, and to manage the end-to-end processing of data. We present a new data scheduling paradigm with advance reservation and provisioning. Our methodology provides a basis for provisioning end-to-end high performance data transfers which require integration between system, storage and network resources, and coordination between reservation managers and data transfer nodes. This allows researchers/users and higher level meta-schedulers to use data placement as a service where they can plan ahead and reserve time and resources for their data movement operations. We present a novel approach for evaluating time-dependent structures with bandwidth guaranteed paths. We present a practical online scheduling model using advance reservation in dynamic network with time constraints. In addition, we report a new polynomial algorithm presenting possible reservation options and alternatives for earliest completion and shortest transfer duration. We enhance the advance network reservation system by extending the underlying mechanism to provide a new service in which users submit their constraints and the system suggests possible reservation requests satisfying users' requirements. We have studied scheduling data transfer operation with resource and time conflicts. We have developed a new scheduling methodology considering resource allocation in client sites and bandwidth allocation on network link connecting resources. Some other major contributions of our study include enhanced reliability, adaptability, and performance optimization of distributed data placement tasks. While designing this new data scheduling architecture, we also developed other important methodologies such as early error detection, failure awareness, job aggregation, and dynamic adaptation of distributed data placement tasks. The adaptive tuning includes dynamically setting data transfer parameters and controlling utilization of available network capacity. Our research aims to provide a middleware to improve the data bottleneck in high performance computing systems.

# Chapter 1

# Introduction

We are witnessing a new era that offers new opportunities to conduct scientific research with the help of recent advancements in computational, storage, and network technologies. Scientific experimental facilities generate massive data sets that need to be transferred to remote collaborating sites. In order to provide high-speed on-demand data access between collaborating institutions, next generation research networks have been deployed for predictable performance and efficient resource utilization. Although through the use of distributed resources the institutions and organizations gain access to the resources needed for their large-scale applications, complex middleware is required to orchestrate the use of these storage and network resources between collaborating parties, and to manage end-to-end processing of the data. In this study, we propose a new data scheduling paradigm in which data movement operations are scheduled in advance with a preferred time constraint given by the user, stating the earliest start time and desired latest completion time. Dealing with particular time windows during scheduling of data transfer operations enables the scheduler to make more accurate decisions satisfying user requirements (such as deadlines, priorities, end-to-end performance, efficiency, and other requirements).

## 1.1 Motivation

Data management has perpetually remained one of the crucial problems in every stage of computing, from micro (CPU chip design) level to macro (Internet and Grid infrastructure) level. Accessing data in a transparent and efficient manner is a major issue, in operating system design, in microprocessors, and in supercomputers [76]. However, other issues come into play in a data transfer paradigm of large scientific data sets between geographically separated resources. Data movement tasks are managed by a separate component in which transfer requests are ordered for better utilization and scheduled

for performance and reliable completion of the given tasks [24]. The dynamic nature of interconnects between collaborating sites, heterogeneity of resources, and client/server side capacity bottlenecks (such as memory, CPU, storage capacity) necessitate provisioning (preparing network, server and client before initiating a data transfer operation for desired transfer throughput) for efficient resource utilization and performance.

A very simple use case can be explained as follows. Consider a scientific application which generates immense amount of simulation data using supercomputing resources. The generated data is stored in a temporary space and need to be moved to a data repository for further processing or archiving. Often, the data repository is located in a remote site where the generated data will be analyzed/visualized by collaborating researchers. In the remote site, another temporary space with limited lifetime may be allocated to store the data. Another application may be waiting this generated data as its input to start execution. We can allocate compute resources in advance, and we even can predict the completion time of a compute job submitted in a supercomputer queue. Therefore, users have the opportunity to have an accurate estimation about the time their computation and analysis will finish, and the generated data will be available to be moved to the remote repository.

In current systems [66, 9, 24, 46], a data transfer request is managed by the scheduler without any constraints. The data transfer request is put in a queue to be scheduled after completing currently running operations. This request may be delayed because of prior long-running jobs, or it can be postponed by the scheduler to operate other short jobs. Depending on the scheduler's policy, the scheduler can initiate other jobs using some of resources shared by this job. In such a case, the number of jobs completed will increase, but the total completion time of our data transfer job will also increase. Delaying the data transfer operation, completing the transfer far after than the expected finish time, may create several problems. One common case is that other resources are allocated for further processing but they are waiting idle for the transfer operation to complete.

Delivering data placement (moving data between collaborating parties) as-a-service where users can schedule their request in advance is highly desirable. In a data placement request, users can pro-

vide a simple time constraint in which they state the earliest start time and latest completion time. Earliest start time specifies when the source data set will be ready to schedule the given task. Latest completion time specifies a desired deadline to complete the transfer operation. The scheduler confirms the request after checking availability of resources and other tasks in the given time frame. If the request cannot be confirmed in the given time frame, the scheduler suggests a longer time period such that latest completion time extended to satisfy the request. It is the scheduler's responsibility to satisfy given requests with the given time constraints. Future time windows are considered while accepting a request and initial decision are made in advance. On the other hand, the scheduler can accept a request that need to be initiated instantly if there is capacity available and none of the reserved operations will be delayed. A transfer operation may come with an on-demand schedule request where it is optimized for earliest completion time.

## 1.2 Methodology

A major challenge is to predict the completion time and also estimate the capacity of a resource involved in data transfer operations. We assume that data scheduler gathers information about the server capacities to control load on data servers in source and destination sites and the number of concurrent tasks operating over those data servers. On the other hand, predicting performance and completion time over a dynamic and shared network is quite difficult [22, 113]. We use a dynamic approach in which transfers operate in best effort to utilize as much bandwidth as possible by adaptively setting the tuning parameters on the fly. The scheduler can keep historic data (min, max and average transfer throughput values between two end-points) to calculate the estimated duration of a transfer while making the scheduling decision. Despite that, the scheduler cannot guarantee the total duration of the transfer; therefore, it tries to schedule the request and initiate the transfer as early as possible to deal with a worst case scenario if happens. In such a case, the scheduler either accepts a given request without confirming the completion time but trying to finish in the desired duration, or confirms the request if a large time window is given that can handle the worst case scenario.

Next generation research networks such as Internet 2 [5] and ESnet (Energy Sciences Network)[3] provide bandwidth guaranteed on-demand data access between collaboration institutions. Advance network reservation systems such as ESNet's OSCARS [8] enable data schedulers to retrieve possible future reservations and allocate bandwidth between two sites for a given duration with predictable throughput. Using a network interconnect in which we can reserve and guarantee bandwidth enables data scheduler to make more accurate decisions and satisfy user requirements with given time constraints.

We propose a data scheduling methodology with advance reservation and provisioning. In order to clarify the concepts explained in this study, we can evaluate the scheduling problem in two separate phases. The first phase is to accept data transfer requests and allocate resources in advance. The scheduler checks the availability of resources in a given time period and justifies whether requested operation can be satisfied with the given time constraints. The server and the network capacity are allocated for the future time period in advance. If there is no available slot to execute the transfer operation, the completion time given by the user is extended and the user is notified about the possible finish time. In this phase, the scheduler instantly searches for availability of resources and tries to fit a request to an empty slot without affecting previously confirmed requests.

The scheduler considers other requests reserved for future time windows and re-orders operations in the current time period. It uses early error detection modules to check resources and data transfer services before confirming a request. In the second phase, we use traditional scheduling techniques in which we organize data transfer request for the current time period. A transfer request may complete much earlier than estimated finish time. There may be a failure and we may or may not retry the operation in the current time window based on the nature of the error and also the deadlines of other waiting jobs. We may also need to release previously allocated resources to make new reservations if possible, if there is available slot to move the job start time backwards. Conversely, data transfer jobs can be moved forward if there are delays while executing previous operations due to some failures or performance degradations. In the second phase, the scheduler operates in an opportunistic manner to maximize resource utilization. However, in the first phase, the scheduler takes into consideration of

worst case scenarios to satisfy deadlines. It allocates resources instantly and for a certain time based on the resource capacity and best-known (historic data) transfer rates.

## 1.3 Contributions

In this dissertation, we present a new data scheduling paradigm with advance reservation and provisioning. Our methodology provides a basis for provisioning end-to-end high performance data transfers which requires integration between storage and network resources, and coordination between reservation managers and data transfer nodes. This allows researchers/users and higher level meta-schedulers to use data placement as a service where they can plan ahead and reserve scheduler time for their data movement operations. Our scheduling framework gives user opportunity to specify their resource and time requirements. We benefit from network provisioning and advanced bandwidth reservation for on-demand high performance data transfers. The new paradigm provided in this study is intended to eliminate possible long delays in completion of a transfer operation by taking advantage of bandwidth guaranteed paths and user defined time constraints, and increase utilization both in client and server sites by giving an opportunity to provision resources in advance. In our approach, we benefit from user-provided parameters of total volume, earliest start and latest completion times. This also enables data schedulers to make better and more precise scheduling decisions by focusing on a particular time period with a number of requests to be organized for given user criteria.

In advance reservation, we deal with dynamic time-dependent networks with guaranteed capacities. Known time-dependent flow algorithms [81, 41, 34] in the literature do not fit into our problem domain. Since we are dealing with bandwidth allocation, we need different types of algorithms to analyze time-dependent network graphs. We present a novel approach for evaluating time-dependent structures with bandwidth guaranteed paths. We discretize dynamic graphs into time steps and time windows and apply known graph algorithms efficiently on static snapshot graphs. We evaluate the complexity of scheduling problem with resource and time conflicts. We present a practical online scheduling model using advance reservation in dynamic network with time constraints.

A major component needed to support the needs of scientific applications is the communication infrastructure which enables high performance visualization, large volume data analysis. Network reservation systems establish guaranteed bandwidth of secure virtual circuits at a certain time, for a certain bandwidth and length of time. If the requested reservation cannot be granted, no further suggestion is returned back to the user. Further, there is no possibility from the user's view-point to make an optimal choice, which also leads to ineffective use of the overall system. We report a new polynomial algorithm presenting possible reservation options and alternatives for earliest completion and shortest transfer duration. We enhance the advance network reservation system by extending the underlying mechanism to provide a new service in which users submit their constraints and the system suggests possible reservation requests satisfying users' requirements.

Some other major contributions of our study include enhanced reliability, adaptability, and performance optimization of scheduling distributed data placement tasks. While designing this new data scheduling architecture, we also developed some important methodologies such as early error detection, failure awareness, job aggregation, and dynamic adaptation of distributed data placement tasks. Early error detection is a major component to check the current conditions of the resources before confirming a given request for scheduling in advance. The scheduler can aggregate tasks and initiate large data movement tasks to increase efficiency and utilization of resources for better performance. The adaptive tuning includes dynamically setting data transfer parameters and controlling utilization of available network capacity.

Our research aims to provide a middleware to improve the data bottleneck in high performance computing systems. We focus on planning and sharing of data for efficient use of large-scale systems in collaborative science. We have explored data-intensive distributed computing and studied challenges in data placement in distributed environments by analyzing key attributes. The algorithm for network reservation presented in this study is developed as a new service extending the underlying mechanism of ESnet's advance network reservation system, OSCARS [8]. Several other contributions such as failure-awareness, and failure recovery, aggregation of data transfer tasks have been applied in the first full release of Stork [9] data transfer scheduler. The dynamic adaptation has also been

implemented and demonstrated to the community as a significant tool for data transfer performance optimization. Results presented in this dissertation also contributes to the research community by analyzing resource allocation and management, fault tolerance in data movement, and coordination of bulk data transfers in data intensive computing.

Many of the extended features we mention throughput the following chapters have already been implemented and tested independently as a part of this study. Those include early error detection modules, error classification and failure aware scheduling, job aggregation, data transfer module with adaptive tuning for best possible bandwidth utilization, and a network reservation engine to suggest possible bandwidth allocations in advance with given constraints. We give detailed analysis of the proposed approaches and we explain directions for future research in the area.

# Chapter 2

# Related Work

Scientific applications especially in areas such as physics, biology, and astronomy have become more complex and compute intensive over the years. Often, such applications require geographically distributed resources to satisfy their immense computational requirements. Consequently, these applications have increasing distributed data intensive requirements, dealing with petabytes of data. The distributed nature of the resources made remote data access and movement the major bottleneck for the end-to-end application performance. Complex middleware is required to orchestrate the use of the storage and network resources between collaborating parties and to manage the end-to-end distribution of data.

## 2.1 Data-Aware Distributed Computing

Recent progress in high performance computing and distributed systems middleware have provided collaborative studies and essential compute-power for science, and made data management a critical challenge in the area. Scientific applications have become more data intensive [55, 79]. The SuperNova project in astronomy is producing terabytes of data per day, and a tremendous increase is expected in the volume of data in the next few years [12]. The LSST (Large Synoptic Survey Telescope) is scanning the sky for transient objects and producing more than ten terabytes of data per simulation [107]. Similarly, simulations in bimolecular engineering generate huge data-sets to be shared between geographically distributed sites. In climate research, data from every measurement and simulation is more than one terabyte. In high-energy physics [32], processing of petabytes of experimental data remains the main problem affecting quality of the real-time decision making.

There are several studies concentrating on data management in scientific applications [13, 14, 109]; however, resource allocation and job scheduling considering the data requirements still remains

as an open problem. Data intensive characteristics of current applications brought a new concept of data-aware scheduling in distributed scientific computation [66, 9].

The concept of data transfer scheduling has been introduced in the literature [24, 76, 13, 14, 109]; however, this is still an emerging field and there are many open problems. Data intensive characteristics of current applications has led us to investigate data-aware resource allocation and scheduling models [109, 85, 86].

Data transfer scheduling is also an important component in workflow management. We can simply classify the steps in a large scale application as follows; (1) obtain data from experiments or simulate to generate data; (2) transfer data and organize for pre-processing; (3) data analysis; (4) move data for post-processing. There have been recent studies for workflow management [72, 103, 104], but providing input/output order for each component, sub-workflows, and performance issues and data-driven flow control are still open for research and development.

There is a few work towards coordinating resource allocation and advance reservation together for data movements [95]. Existing systems fail to address issues such as scheduling according to given user requirements and priorities, taking advantage of advance resource reservation, and adapting to dynamic environment in distributed systems. Current data schedulers manage data transfer jobs by trying to optimize for performance and resource utilization [66, 9, 24, 46], but they do not provide advance resource reservation and coordination where users can plan ahead and allocate/reserve the data placement service for a future time.

## 2.2 High-Performance Data Transfers

There have been studies for high throughput data transfer in which best possible paths between remote sites are chosen and servers are optimized to gain high data transfer rates [24]. There are also various specialized TCP protocols [45, 98, 100] which change TCP features for large data transfers. There have been studies for fast file downloading [83]; and tools have been developed to measure

the network metrics [33, 77] to tune TCP by setting the window and buffer size for optimum performance. Characteristics of the communication structure determine which action should be taken when tuning the data transfer operation. Local area networks and wide area networks have different characteristics, so they demonstrate diverse features in terms of congestion, failure rate, and latency. Dedicated channels such as fiber-optic networks require special management techniques [100]. There are high-speed network structures with channel reservation strategies for fast and reliable communication. Congestion is not a concern in such reserved network channels. Application and storage layers should feed the network with enough data in order to obtain high throughput with maximum utilization. We deal with a shared environment and network conditions may change over time. The achievable end-to-end throughput and the system load in communicating parties might alter during the period of a data transfer especially in which large volume of data needs to be transmitted. Therefore, dynamic tuning approaches that adapt underlying network layer and to optimize data movement accordingly has been developed [22]. Furthermore, server capacity has also crucial role to identify how much data can be sent or received in a given interval. Therefore, storage systems have been developed for efficient movement of data [60, 67] by providing caching mechanism and multi-threaded processing. Data servers are distributed on different locations and available network is usually shared (i.e. the Internet); therefore, minimizing the network cost by selecting the path which gives maximum bandwidth and minimum network delay to obtain high speed transfer will increase the overall throughput [15, 78]. We can measure the values of attributes like CPU load, memory usage, and available disk space on server side. On the other hand, we usually utilize predicted values for network features which are calculated according to previous states. Since each data movement job is competing with each other, a decision making is required to have them ordered and run concurrently. We may need to decline an upload operation till data files are downloaded so that there is available space in the storage. Moreover, we might delay a data transfer job if network is under heavy utilization due to some other data transfer operations.

Transfers especially over wide-area networks encounter different problems and should deal with obtaining sufficient bandwidth, dealing with network errors and allocating space both in destination

and source. There are similar approaches to make resources usable to science community [47, 106, 30]. Replica catalogs, storage management and data movement tools are addressing some of those issues in the middleware technology [52, 73, 39].

## 2.3 Data Transfer Scheduling

Due to the nature of distributed environments, the underlying infrastructure needs to be managing the dynamic behavior of heterogeneous systems, communication overhead, resource utilization, location transparency and data migration. One important challenge is to manage storage space in data servers. Input data should be staged-in and generated output should be staged-out after the completion of the job. Insufficient storage space will delay the execution, or the running task may crash due to improperly managing storage space in the server. Some storage servers enable users to allocate space before submitting the job and they publish status information such as available storage space. Different techniques such as smallest fit, best fit, and largest fit have been studied [65, 66]. Performances of the file systems, network protocol, concurrent and parallel transfers are some examples influencing the performance of data movement. Another important feature is fault tolerant transfer of data objects. Storage servers may create problems due to too many concurrent write requests; data may be corrupted because of a faulty hardware; transfer can hang without any acknowledgment. Access and failure transparencies are other major issues such that user tasks should access resources in a standard way and complete execution without being affected by dynamic changes in the overall distributed system [23, 18].

Simulating the Data Grid to investigate the behavior of various allocation approaches have been studied [87, 80, 68]. Beside the simple greedy scheduling techniques such as Least Frequently Used (LFU) and Least Recently Used (LRU), there are also some economical models handling data management and task allocation as a whole while making the scheduling decision [109]. Another recent work concludes that allocating resources closest to the data required gives the best scheduling strategy [85, 86]. There are many studies on replica management, high performance data transfer, and data

storage organization; however, there is still a gap in data-aware scheduling satisfying requirements of current e-Science applications. One common scenario is to transfer large amount of data to a central data center from several geographically distributed resources. In [16], authors study provisioning of data aggregation sessions and propose a heuristic to schedule data aggregation requests in which multiple files from multiple data resources are transferred to a single destination. The problem introduced in [16] focuses on data aggregation from multiple data centers in lamda grids, such as National LamdaRail [7]. Since we deal with a dynamic environment, provisioning of data aggregation requests are accomplished on demand. The proposed solution aims to provision multiple file transfer requests while maximizing the bandwidth in which the file transfer rates between two end-points are calculated considering the heterogeneity of server resources. One interesting objective explained in this study is to try to combine requests and take as a whole to minimize number of used resources; such that the number of resources not allocated and ready for the future transfer request will be maximized.

In [48], a reservation and allocation architecture, GARA(Globus Architecture for Reservation and Allocation), is defined to address several problems in providing end-to-end quality of service for next generation research networks. Heterogeneity of resources requires independent control and local administration policies of individual resources. Computational elements also affect end-to-end performance and they should be managed and monitored separately while dealing with reservation elements. The GARA project aims to provide application level co-allocation by providing a reservation API in order to coordinate resources, and to allocate them in advance.

There are also several relevant studies in the literature using reinforcement learning for resource management and planning [49], and user constraints for file transfer scheduling [31]. Priority-based scheduling has been studied for real-time system [90], especially for databases to satisfy time constraints with transactions [40]. Deadline scheduling algorithms consider the time constraint of every request to ensure the deadline (completion time). We can classify real-time database transaction into two categories; hard and soft transactions. In a hard real-time transaction system, the scheduler needs to guarantee the completion time. There is no benefit to finish the request after the deadline. In a soft real-time transaction system, the scheduler considers the time constraint and prioritizes the requests

with earliest deadline in the scheduling queue [90]. In our data scheduling paradigm, we consider soft-deadline scheduling. We can allocate the server and the network capacity (bandwidth); however, it is difficult to guarantee the completion time of a data transfer request. Therefore, there are two objectives in our scheduling mechanism. The data scheduler takes into account completion time constraints while making the decision to maximize resource utilization.

## 2.4 Resource Reservation for Guaranteed-Bandwidth

The need for transferring data chunks of ever-increasing sizes through the network shows no sign of abating. A major component needed to support the current data-intensive applications is the communication infrastructure that enables high performance visualization, large volume data analysis, and also provides access to computational resources. There is an increase in developing projects for research networks to provide dedicated bandwidth channels. The dedicated bandwidth networks brings the ability to provision the communication channels when the data, especially large-scale massive data, is ready to be transferred [69, 88].

Delivering network-as-a-service that provides predictable performance, efficient resource utilization and better coordination between compute and storage resources is highly desirable. In order to provide high-speed on-demand data access between collaborating institutions, research institutions established production level network supporting on-demand bandwidth reservation in which bandwidth is reserved for a specific time period [8, 5, 4]. On-demand bandwidth reservation is usually supported by Multiple Protocol Label Switching (MPLS) in layer 3 [27, 42]. In layer 2, a virtual secure circuit is setup between source and destination with a specific bandwidth over the connection. There are few studies in on-demand bandwidth allocation [1, 8, 2] and advance bandwidth reservation [53, 89, 29, 91, 58]. A very typical case is to represent the network topology as a graph. In addition to that, we need a proper representation for time in advance reservation. There are two common approaches; slotted time model and continuous time model. In slotted time model, time is divided into equal slots and each link keeps information about the available bandwidth in each slot. Several

studies addressing slotted time model are [53, 29, 102, 92, 48]. In continuous time model, the link capacity is modeled as a time-bandwidth function. This provides better granularity and finer control in scheduling with a cost of increased complexity in implementation. Several studies addressing contiguous time model include [91, 114, 89].

Internet 2 [5] and ESnet (Energy Sciences Network) [3] are some of the well known efforts to support next generation research networks. The Energy Sciences Network (ESnet) provides high bandwidth connections between research laboratories and academic institutions for data sharing and video/voice communication. The ESnet On-Demand Secure Circuits and Advance Reservation System (OSCARS) establishes guaranteed bandwidth of secure virtual circuits at a certain time, for a certain bandwidth and length of time. Though OSCARS operates within the ESnet, it also supplies end-to-end provisioning between multiple autonomous network domains. OSCARS gets reservation requests through a standard web service interface, and conducts a Quality-of-service (QoS) path for bandwidth guarantees. Multi-protocol Label Switching (MPLS) and the Resource Reservation Protocol (RSVP) enable to create a virtual circuit using Label Switched Paths (LSP's). It contains three main components: a reservation manager, a bandwidth scheduler, and a path setup subsystem [54, 8]. The bandwidth scheduler needs to have information about the current and future states of the network topology in order to accomplish end-to-end bandwidth guaranteed paths.

In [91, 58], path computation for advance bandwidth reservation problem has been analyzed and some approaches in the literature are evaluated. Some of the solutions for advance bandwidth reservation are categorized into the following problem domains [91, 58]. First case is to reserve a fixed slot in which we find a path from source to destination with a specific bandwidth value (a specified bandwidth in a specific time slot). Second is to find the path with largest bandwidth from source to destination in a given time period (highest available bandwidth in a specific time slot). Third is to find a path that gives maximum duration of allocation with a specific bandwidth value starting at a specific time. And other cases are to find first or all slots in which there is a path with the specific bandwidth from source to destination (time slots with specified bandwidth and duration). All those problem cases can be solved by extending very well known graph algorithms such as breadth-first

search, Dijkstra, and Floyd algorithms [91]. Even more, the proposed solution methods in [91, 58] are based on slotted time window model. The network topology is represented as a graph with time information on it such that each link has the available bandwidth in every time slot.

The given problem cases in [91, 58] do not address the scheduling dilemma for resource allocation and reservation in high performance data transfers. Instead, given solutions (path with the highest available bandwidth in a given time period, all available time slots with a specific bandwidth value, etc.) are modified versions of know algorithms in a time-dependent graph structure. They do not compute an optimal reservation for a massive data transfer request, and do not suggest any allocation pattern. Therefore, we necessitate a new paradigm and solution style for advance reservation to define the methodology and to provide the solution incorporated with other resource management issues to address high performance data transfer for massive data sets. Furthermore, the given algorithms in [91, 58] have high complexity and large space requirements. The proposed approach should be easily applicable and very efficient for real-life advance reservation systems.

## 2.5 Advance Reservation and Time Constraints

In advance network reservation, we first need to ensure the availability of requested bandwidth before committing a bandwidth allocation request. The foremost question is how to find the maximum bandwidth available for allocation from a source node $n_{source}$ to a destination node $n_{destination}$. The max-bandwidth path algorithm is well known in quality-of-service (QoS) routing problems in which a path is constructed from source to destination given that each link is associated with an available bandwidth value. The bandwidth of a path is the minimum of all links over the path. In max-bandwidth algorithm, we find a path from source $n_{source}$ to destination $n_{destination}$ whose bandwidth is maximized. The bandwidth of a path from $n_A$ to $n_C$ which is constructed by adding link $e^l : B \rightarrow C$ is $min\{b_{bandwidth}(e^l), B_{max-bandwidth}(A \rightarrow C)\}$. The QoS condition is a bottleneck constraint in max-bandwidth path calculation. Alternatively, in shortest path calculation, we find a path whose sum of weights is minimized, and QoS constraint is additive (minimum delay path, or minimum hop

count path).The max-bandwidth path algorithm is a slightly modified version of Kruskal and Dijkstra's algorithms with the same asymmetrical time complexity [82]. Alternatively, QoS constraint is additive in shortest path calculation (minimum delay path, or minimum hop count path).In shortest path algorithm, we find a path whose sum of weights is minimized. The main objective in shortest path can be to minimize the hop count, or "engineering metric" which is associated with latency in our graph representation. In the shortest path algorithm, the weight of a path is the sum of values added by each link in the path. On the other hand, the weight of a path in max-bandwidth is the minimum link bandwidth, the bottleneck link over the path.

Those algorithms are very fast and efficient, and they have been adapted to deal with many problems in routing and gateway protocols. Shortest path, min-cost path, max-bandwidth path, and minimum spanning tree algorithms can be implemented using a very similar methodology with simple adjustments. In a graph with $n$ nodes, there is a total $n!$ paths from source to destination. The main advantage of those types of graph algorithms is that maximum $n^2$ paths are visited even in worst case. In addition to typical graph representation of network topology, we also need a proper structure for "time" in advance reservation.

We deal with a dynamic network such that the bandwidth value for every link is time dependent. While constructing a path and calculating the available bandwidth over a path, we need to consider another variable, time; therefore, the dimension of the problem is extended by adding the time variable such that the state of the topology depends on the time period. Graph algorithms for time-dependent dynamic networks has been studied in the literature especially for max-flow and shortest path algorithms [81, 41, 34]. The most common approach is the discrete-time algorithms in which the time is modeled as a set of discrete values and a static graph is constructed for every time interval. As an example, [36] uses time-expanded max flow for data transfer scheduling, and [81] presents various shortest path algorithms for dynamic networks with time-dependent edge weights.

**Analogous Example:** We need different types of algorithms to analyze time-dependent max-bandwidth path calculation. The following is given to clarify the advance bandwidth reservation in

dynamic networks. Assume a vehicle wants to travel from city $A$ to city $B$ where there are multiple cities between $A$ and $B$ connected with separate highways. Each highway has a specific speed limit but we need to reduce our speed if there is high traffic load on the road, and we know the load on each highway for every time period.

The first question is which path the vehicle should follow in order to reach city $B$ as early as possible. Alternatively, we can delay our journey and start later if the total travel time would be reduced. Thus, the second question is to find the route along with the starting time for shortest travel duration.

Time-dependent graph algorithms mainly focus on those two questions. However, we are dealing with bandwidth reservation where allocation should be set in advance when a request is received. If we apply this condition to the example problem described above, we have to set the speed limit before starting and cannot change that during the journey.Therefore, known algorithms do not fit into our problem domain. This distinguishes our path calculation from other time-dependent graph algorithms in the literature.

Bandwidth scheduling for multiple data transfer has been studies in [70]. Two different scheduling scenarios are considered. The first one is to schedule multiple data transfer request over some predetermined paths while minimizing total completion time. The second scenario in [70], a more interesting problem, is to schedule multiple bandwidth reservation requests each with a specified time slot. The first problem can be solved with an optimum algorithm. The idea is to schedule requests with less data by evenly distributing them over multiple paths. Considering jobs shorter in duration first minimizes the total waiting time, so reduces the total completion time. The second problem is to assign a network path for each reservation request with fixed bandwidth and predetermined (fixed) time period. The question is to come up with an order and preference to allocate link bandwidths while maximizing the total number of reservation request satisfied. However, this second problem is complex (NP-complete); therefore, a greedy heuristic is given [70]. A similar approach is used in which requests consuming less resource are given preference in scheduling. Resource usage is repre-

sented by a bandwidth distance product (distance is computed by hop count in a possible path from source to destination) which is calculated for each reservation request; and, the requests with small bandwidth distance product are considered first.

In [70], the proposed scheduling algorithms are classified as periodic scheduling and differentiated from instant scheduling. In instant scheduling, the scheduler makes a decision for every incoming request. In periodic scheduling, the scheduler makes decision in certain intervals where several requests in that period are considered. There are multiple requests in the system and scheduler finds an assignment for each request according to some criteria. However, most of the proposed approaches in the literature are for instant scheduling algorithms. In [38], the scheduler algorithm considers flexibility in bandwidth and the time period in order to increase utilization. [108, 50] introduces a new concept by using varying bandwidth over multiple time slots to minimize the completion time. In [71], authors address possible scheduling problems for combinations of four cases; variable bandwidth and fixed bandwidth, and variable path and fixed path.

# Chapter 3

# Data Scheduling in Distributed Environments

Data management has perpetually remained one of the crucial problems in every stage of computing, from micro (CPU chip design) level to macro (Internet and Grid infrastructure) level [76]. We have studied traditional data transfer paradigms in operating systems and microprocessor architectures, while aiming to derive from data transfer techniques used in these micro-systems. After analyzing different application scenarios, we have developed new data scheduling methodologies and the key attributes for reliability, adaptability and performance optimization of scheduling distributed data placement tasks. Three major contributions of this work include (i) adaptive scheduling of data placement tasks for improved end-to-end performance, (ii) aggregation of data placement jobs for increased data transfer throughput, and (iii) a failure-aware data placement paradigm.

Accessing data in a transparent and efficient manner is a major issue, in operating system design, in microprocessors, and in supercomputers [1]. In operating systems, efficiently moving pages from disk to memory is crucial; in microprocessor architecture, instruction fetch time plays an important role. On large-scale distributed systems, transferring data files between geographically-separated storage sites, and optimizing data access in supercomputers, have major effects on overall performance. In our recent study [76], we have presented a generic data management model by looking for similar analogies in different layers of computing systems ranging microprocessor to distributed systems. We study how challenges encountered in the early phases of computers can be addressed in distributed systems to provide a broader perspective in which we can extend known methodologies.

All computing systems consist of three core components: (i) storage from where to read/write data, (ii) an arithmetic logic unit for computing (or modifying data), and (iii) a bus to transmit data between the computation unit and storage. Likewise, distributed systems are made up of data storage

[1] Ken Batcher's half-serious, half-humorous Supercomputer definition interestingly addresses the importance of data placement middleware: "A supercomputer is a device for turning compute-bound problems into I/O-bound problems"

elements, computational nodes, and network interconnects to transmit data between computational nodes and storage elements. In microprocessors, most instructions can be classified into either arithmetic instructions or memory access (e.g. load/store) instructions. Similarly, operations in a distributed data-aware systems are classified into computational tasks and data access/movement tasks [66, 24]. In microprocessors data retrieval from memory to the execution unit is slow due to the speed of memory and also the latency in the bus between memory and CPU. Multi-level caches, dynamic scheduling of I/O operations, and complex branch prediction algorithms are some techniques used to hide memory latency. At the operating system level, many strategies are used to access data efficiently, including DMA, I/O scheduling, and low-level parallelism such as disk striping. Similarly, in distributed systems, we organize and schedule data placement tasks out-of-order to improve the utilization of underlying storage resources. Also, we enhance data transfer throughput by aggregating and optimizing data transfer operations to hide the network latency. Figure 3 gives the overall picture representing data handling in these different layers.

In spite of the above similarities, data placement scenario in distributed environment is different from microprocessors and operating systems in terms of the dynamic nature of interconnects between resources. While, data transfer within microprocessors and in operating systems is done through high speed bus and is deterministic in most cases, data movement in a distributed system operates in a dynamic environment. It is prone to frequent failures resulting from back-end system level problems. Hence, we use a dynamic network where data placement middleware needs to adapt to the changing conditions in the environment. Therefore, we have studied adaptive scheduling of data placement tasks for high performance data transfer. The adaptive scheduling approach includes dynamically tuning data transfer parameters over wide area networks for efficient utilization of available network capacity and optimized end-to-end data transfer performance. Inspired by prefetching and caching techniques in microprocessors, we have implemented aggregation of data transfer requests in order to increase the throughput especially for transfers of small data files by minimizing the effect of connection and protocol setup time. We have presented a failure-aware data placement paradigm for increased fault-tolerance. The failure-aware data placement includes early error detection, error

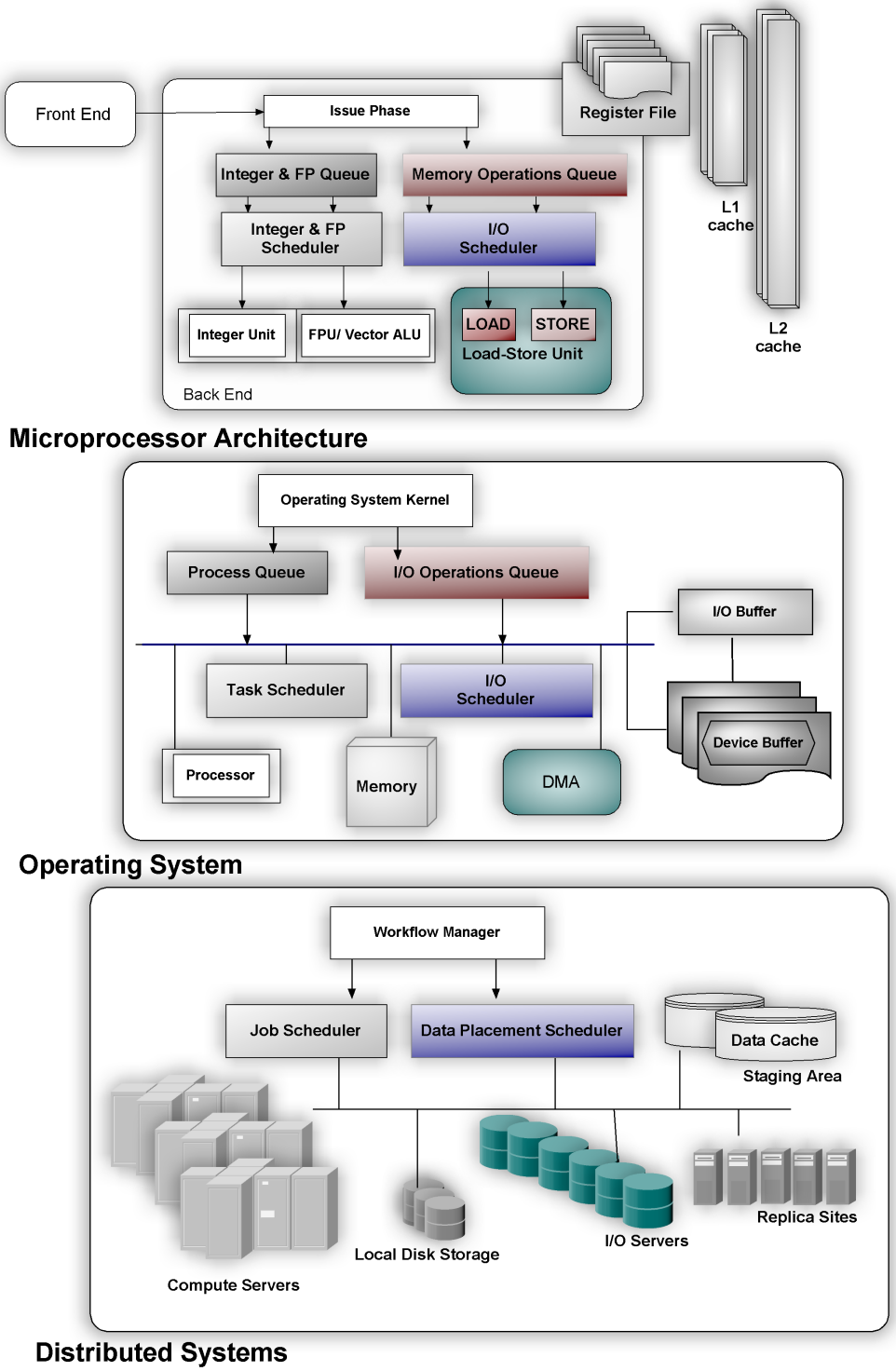



Figure 3.1: Different layers of computing systems illustrating similarities in data and computation management subsystems

classification, and use of this information in scheduling decisions for the prevention of and recovery from possible future errors.

## 3.1 Adaptive Tuning

Similar to memory wall in microprocessors, we see latency wall in data access over high bandwidth connections. We use multiple data streams for fetching data in order to fully utilize the network bandwidth. Using multiple parallel streams in data transfer is simply aggregation of network connections. Instead of a single connection at a time, multiple streams are opened to a single data transfer service in the destination host. We gain larger bandwidth in a network with less packet loss rate; parallel connections better utilize the network buffer available to the data transfer. Excessive use of system resources (CPU, memory, network) may lead to some problems like resource starvation. So, we need to adjust the level of parallelism according to the capacity of the environment. We focus on the level of parallelism in two stages: (1) the number of parallel data streams connected to a data transfer service for increasing the utilization of network bandwidth, and (2) the number of concurrent data transfer operations that are initiated at the same time for better utilization of system resources.

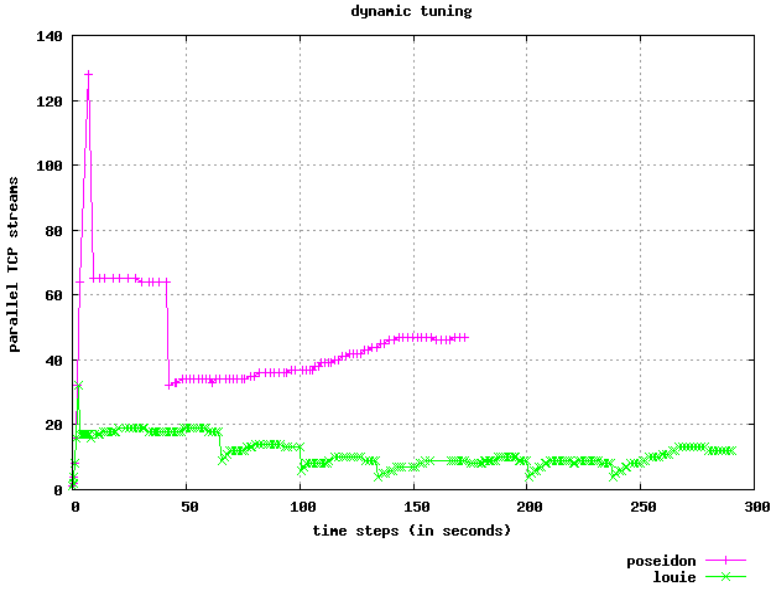
The Stork data scheduler [66, 9, 64] has a modular architecture where data placement jobs are executed by specific data transfer modules according to the protocol specifications of each request. The scheduler accepts multiple jobs in a nondeterministic order, say: $< J1, J2, J3, ... >$. The goal of the scheduler is to minimize the execution time for each data transfer while preserving the user fairness based on submission order. Completion time $T$ of a job depends on environment conditions such as network and system load. Therefore, we try to find the best possible settings, $T =< J, c, p >$, for $c$ (concurrent jobs) and $p$ (parallel streams) in an adaptive manner. Instead of probing the system to get profiling information, we just use performance metrics from actual data transfers for parameter tuning. The scheduler's decision adapts itself to the environment conditions. Gradually improving parallelism level brings a near optimal value without the burden of dealing with complex optimization steps to find the major bottleneck in a data transfer. Impacts of parallel streams as well as concurrent data transfer jobs running simultaneously have been explained in [22]. Our adaptive tuning algorithm dynamically sets the parallelism level [18]. Figure 3.2 shows an illustration of dynamic parameter tuning in which system detects a change in the environment and adjust the number of parallel streams.

Instead of making measurements with external profiler to set the level of concurrency and parallelism in data transfer scheduling, we propose to calculate parameters using information from current running data transfer operations. Thus, we do not pollute the network with extra packets and do not put extra load to the system due to extraneous calculations for exact parameter settings. We simply set the level of parallelism dynamically by observing the achieved application throughput for each transfer operations and gradually tune parameters according current performance merit. Our dynamic tuning algorithm enables to transfer data by chunks and also set control parameters on the fly. It measures the transfer time of each chunk transferred and calculates the current throughput. We keep the record of best throughput for the current parallelism level. The actual throughput value of the data chunk transferred is calculated and the number of parallel streams is increased if this throughput value is larger than the best throughput seen so far. We gradually increase the number of parallel streams till it comes to an equilibrium point.

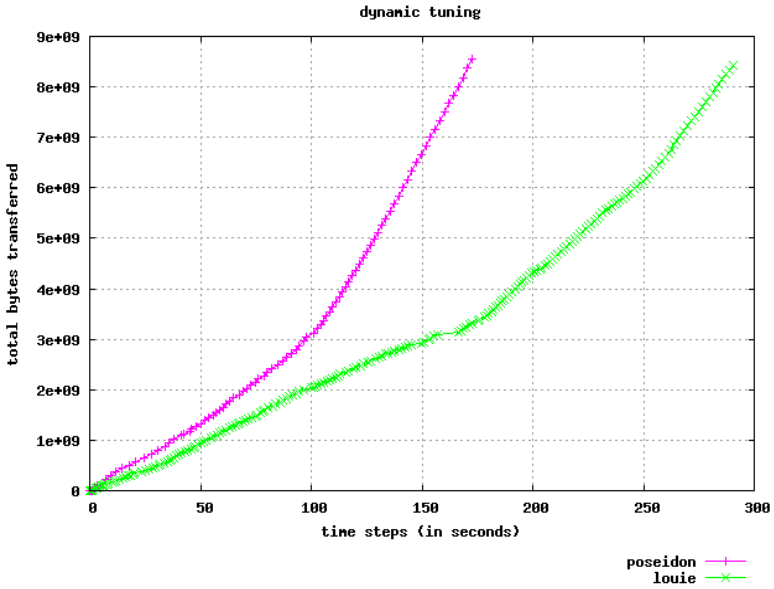
## 3.2 Aggregation of Data Placement Jobs

Completion time of a job also depends on environment conditions such as network and system load. Each transfer operation spends time for protocol initialization and connection setup over the data transfer service. A single transfer $t$ consists of $t_{setup}$ in which we initialize the transfer module and prepare a connection to transfer the data, and $t_{transfer}$ in which we transfer data using the data transfer protocol. Although this connection time is really small and sometimes negligible according to the total duration spent for transferring the data, it plays important roles where there are hundreds of jobs waiting to be scheduled. We target on minimizing the load put by $t_{setup}$; as an example, instead of having two separate operations $t_1 = t_{setup} + t_{1transfer}$ and $t_2 = t_{setup} + t_{2transfer}$, we aggregate request to improve the total transfer time $t = t_{setup} + t_{1transfer} + t_{2transfer}$.

Beyond that, each data transfer operation need to be operated separately by executing a specific data transfer module. Aggregating data placement jobs and combining data transfer request into a single operation also has its benefits in terms of improving the overall scheduler performance. Such




(a) number of parallel streams over time




(b) total bytes transferred over time

Figure 3.2: Adaptive Tuning Algorithm: setting number of parallel streams dynamically for transfers from poseidon and louie to queenbee machines on LONI network [6].

that, this will reduce the total number of requests that data scheduler needs to execute.

There are two major key attributes affecting the completion time $T$: (1) the number of parallel streams to fetch data from destination host, and (2) the number of concurrent jobs scheduled to access

resources of the source and destination hosts. The goal of the scheduler is to minimize the execution time for each data transfer while preserving the user fairness based on submission order. Therefore, we try to find the best possible settings, $T =< J, c, p >$, for $c$ (concurrent jobs) and $p$ (parallel streams). Here, we add another parameter which is the aggregation count, $a$. Therefore, $T =< J, c, p, a >$.

We have applied job aggregation in Stork data placement scheduler such that total throughput is increased by reducing the number of transfer operations. According to the file size and source/destination pairs, data placement jobs are combined and processed as a single transfer job. Information about the aggregated job is stored in the job queue and it is tied to a main job which is actually performing the transfer operation such that it can be queried and reported separately. We have seen vast performance improvement, especially with small data files, simply by combining data placement jobs based on their source or destination addresses [18]. We have tried several ways for job aggregation and we saw increase in total throughput of data transferred even by simply combining data placement jobs based on their source or destination host. Figure 3.3 shows performance of aggregation in data transfer scheduling. The main performance gain comes from decreasing the amount of protocol usage and reducing the number of independent network connections. Aggregation count is the maximum number of requests combined into a single transfer operation. Multiple streams is the number of parallel streams used for a single transfer operation. And, parallel jobs represents the number of simultaneous/concurrent jobs running at the same time. We analyze effects of those parameters over total transfer time of the test-set.

We believe that a pre-processing layer in the data placement scheduler will help us to reorganize jobs and make better scheduling decisions. We do analyze the set of requests in the queue to aggregate data placement tasks. As an example, instead of initiating each request one by one, it is more beneficial to execute them as a single operation if they are requesting data from same storage site or using same protocol to access data. However, contrary to the job aggregation, we might also decompose jobs in the pre-processing layer. This type of pre-processing is similar to the logic behind micro-instructions in processor design. A machine code from a complex instruction set architecture is decomposed into micro-code operations [99]. Especially for non-RISC based microprocessors, we

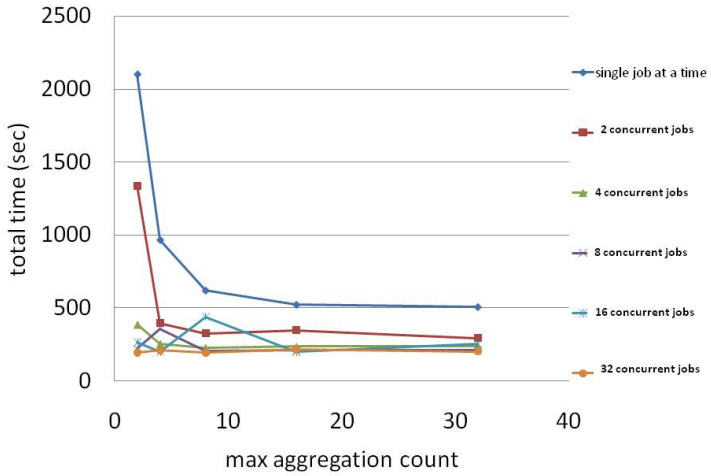
fully benefit from pipelined architecture by the help of micro-instructions. As an example, a set of files to be transferred to or from a remote site can be fetched from multiple replicas located in different storage servers. Thus, we can decompose the data placement request into several small set of file requests and process each simultaneously by using different set of resources. Even for large files, we can initiate partial transfers from multiple servers in which we are fetching data chunks in parallel.

## 3.3 Reliability and Failure-awareness

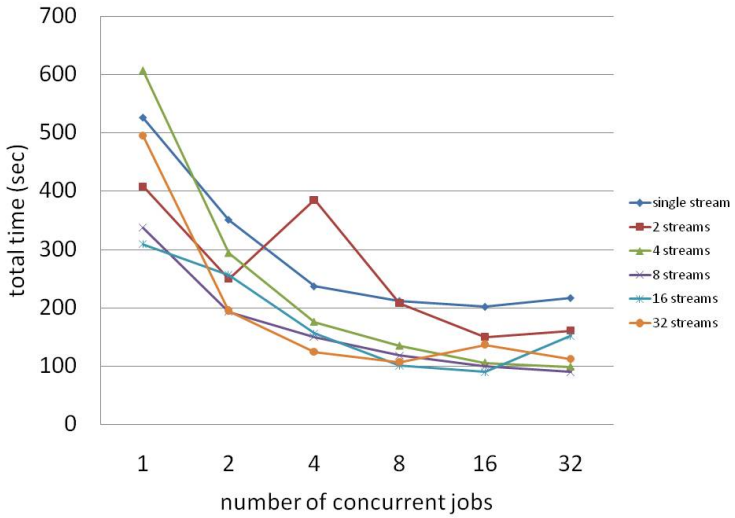
Although latency and throughput are the main performance factors of data transfers (both in highly distributed and closely coupled environments), usability and efficiency of distributed data transfers also depend on some other aspects such as error detection and error reporting. Failure during data transfer in distributed environment is quite common. The major drawback in distributed data transfer is that the user sometimes is not aware of technical facts like the backend network connectivity failures. In most cases the users do not have enough information to infer what went wrong during data transfer because they do not have access to the remote resources, or messages got lost due to system malfunction. Tracking the problem and reporting it back correctly to the user is important to give user a sense of a consistent system.

The problem that we should mitigate first is the lack of sufficient information to clarify the reasons for a failed data transfer. Our study has two main aspects: error detection and error classification. In error detection, we focus on making data placement scheduler aware of whether destination host/service is available, and also making the scheduler able to select suitable data placement transfer services. In error classification, we propose an elaborate error reporting framework to clarify and distinguish failures with possible reasons. Moreover, we discuss the progress cycle of a data transfer operation in which several steps are examined before actually starting the data transmission operation.

Before initiating a data transfer operation in which source host will connect a file transfer service running on a remote server and transmit data over a network channel, it is important to get prior knowledge in order to decrease error detection time. In addition, it is also useful at the time of




(a) aggregation count vs. number of concurrent jobs for transfers over a single stream




(b) number of concurrent jobs vs. number of parallel streams (at most 16 jobs are aggregated)

Figure 3.3: Performance measurement for job aggregation: 1024 transfer jobs from ducky to queenbee machines (rrt avg 5.129ms) on LONI [6] network with 5MB data file per job

scheduling to know whether destination host and service is available or not; such that, a data transfer job which would fail because destination host or service is not reachable, will not be processed until that error condition is recovered. In addition to the advantage of prior error detection, information

about active services in the target machine would help data placement scheduler discover and use alternative transfer protocol.

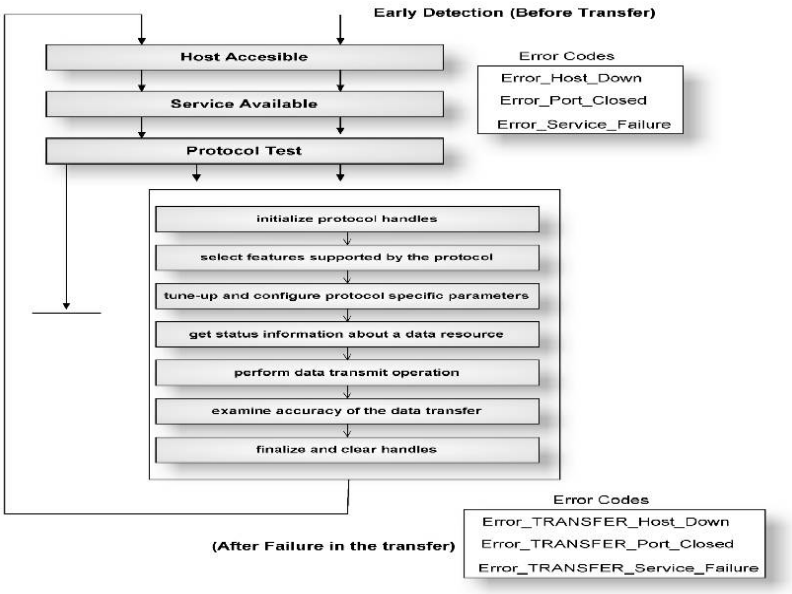
The main purpose of classifying data transfer operation in several categories is to better understand at which stage an error has occurred. File transfer protocol such as GridFtp [10], will generate error codes and error messages. However, proposed error reporting framework will help both users and higher level planners to recognize the error condition such that in respect of the stage where error occurred different actions can be taken. As an example, a directory transfer operation can fail since used file transfer protocol is not supporting directory listing. In such a case, error will fail in the status phase before proceeding to the transmit phase. Therefore, we can provide a better logging facility which can be parsed and used by a higher level planner to get information in which stage operation failed. Besides, we can also understand in which point an error has occurred in each stage. In order to capture errors caused by network failures or mal-functionality in the protocol, we keep state information in every phase. If we get an error after a file transfer operation has already been initiated and data transmission is started for processing, we treat the problem according to the fact that a problem may occurred in the network or remote site. Therefore, we define the operation object with three state types to keep track of status information between each phase. Figure 3.4.a shows codes in error classification for data transfer scheduling.

Prior knowledge about the environment, and awareness of the actual reason behind a failure, would enable data placement scheduler to make better and accurate decisions. Network exploration techniques have been studied in order to classify and detect network error as early as possible [23]. We have also successfully developed error detection and classification strategies for failure-aware data scheduling. The data scheduler checks the network connection and availability of data transfer protocol beforehand, with the help of a new network exploration module. These features also enable us to select amongst the available data transfer services provided by a storage site.

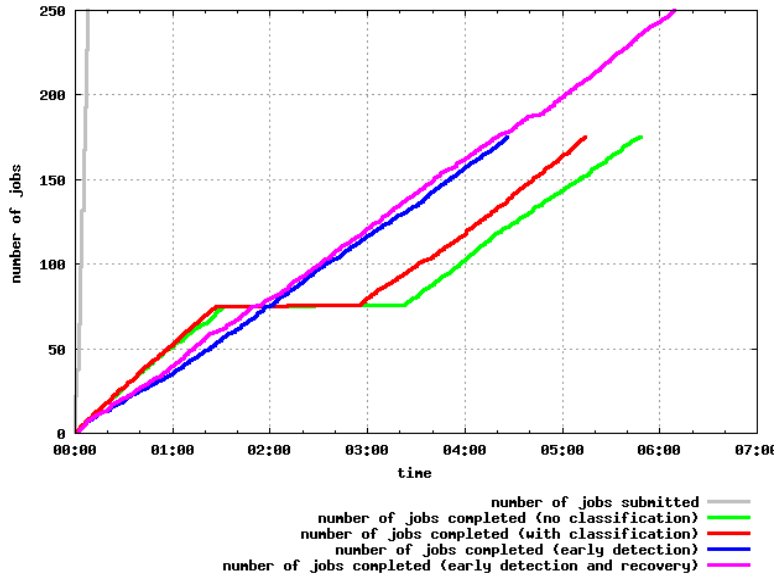
We have experimented impact of error detection and classification in data transfer scheduling. We used 250 data transfer jobs submitted to the Stork scheduler and injected different types of errors

into the system while scheduler is performing given requests. Simply, we change the permission of target directories, forced certification to be expired; such that, the problem in the data transfer occurs because of misconfiguration or improper settings of input output parameters. Besides, there are other types of errors due to server or network outages which can or can not be recovered later. We measure the makespan for all jobs in the system with error classification and without error classification. As expected, scheduler does better decision and do not retry failed jobs if erroneous case cannot be recovered. Results presented in Figure 3.4.b show a heavily loaded queue with all data transfer jobs are submitted in the beginning. It takes longer to complete all jobs when there is no classification, since scheduler retries the failed jobs assuming they can be recovered in the next run. With error classification, failed jobs are classified according to the error states where problems occur, so we do not retry every failed operation. Early error detection feature provides fully classification and data transfer jobs that will fail are detected in advance, so those jobs are not scheduled at all. However, the condition leading to failure may disappear later. Failures are detected beforehand and those jobs are scheduled when the problematic condition has been resolved. Therefore, we see almost the same performance with early detection and recovery if compared to the case without any failure.

Stork, data placement scheduler, checks network connection and availability of the data transfer protocol. Early error detection has been implemented as a new feature inside Stork. Moreover, we propose a generic framework such that error classification is not limited by GridFtp operations. We have also been testing our model with other data transfer protocols like iRods [93]. The GridFtp transfer module in Stork is also able to verify the successful completion of the operation by controlling checksum of each file. Moreover, it can recover from a failed operation by restarting failed data transfer operations. The rescue file keeps track of failed and succeeded file transfer operations. In case of a retry from a failure, the scheduler informs the transfer module to recover and restart the transfer using the information from a rescue file created by the checkpoint-enabled transfer module.




(a) data transfer progress cycle and error classification




(b) performance effect of error detection and classification

Figure 3.4: Experiments with error detection and classification: 250 data transfer jobs were submitted to Stork scheduler and erroneous conditions were injected into the system

# Chapter 4

# Network Reservation Engine

We study network provisioning and advanced bandwidth reservation in ESnet [3] for on-demand high performance data transfers. A reservation request from a user includes desired bandwidth allocation between end-points with duration and starting time information. A bandwidth reservation system, called On-demand Secure Circuits and Advance Reservation System (OSCARS) [8, 54], checks network availability and capacity for the specified duration of time, and allocates it for the user if it is available. Otherwise, it reports to the user that it is unable to provide the required allocation. Accordingly, the user needs to search for a time-frame of a required bandwidth by trial-and-error, not having knowledge of the network's available capacity at a certain instant of time. We improve the current ESnet advance network reservation system, OSCARS, by presenting the clients possible reservation options and alternatives for earliest completion time and shortest transfer duration.

We design an algorithm, where the user specifies the total volume that needs to be transferred, a maximum bandwidth that can be used and provisioned in the client sites, and a desired time window within which the transfer should be done. The proposed algorithm can find alternate allocation possibilities, including earliest time for completion, or shortest transfer duration - leaving the choice to the user. It is quite practical when applied to large networks with hundreds, even thousands of routers and links. We have implemented our algorithm for testing and incorporation into a future version of OSCARS.

In our approach, we discretize the time-dependent dynamic network topology by dividing the search interval into time steps. Each time step represents a stable status of the topology. We provide a methodology to calculate static snapshot graphs in each time step and apply max-bandwidth algorithm while traversing over the search interval. We show that the number of subsequent combinations of time steps, or the number of time windows, is bounded by the number of reservations in

the system. Searching the given time interval is accomplished in polynomial time. Hence, we provide an efficient algorithm to find possible advance network reservation options for the given data transfer requirements.

## 4.1 Advance Network Reservation

The OSCARS bandwidth reservation system keeps track of changes in the network status and maintains a topology graph $G$ which can simply be described as follows. The network topology graph in OSCARS includes routers, ports, and unidirectional links between two ports, $G =< n_{router}, v_{port}, e_{link} >$. Each router has a list of attached ports, $n_{router} =< v^1_{port}, v^2_{port}, .. >$, and each port has a maximum available bandwidth ready for advance allocation. A link connects two ports in one direction, $e^1_{link} =< v^1_{port}, v^2_{port} >$ , $e^2_{link} =< v^2_{port}, v^1_{port} >$; such that, and a separate reservation request is established for each direction. Every port in a router has a maximum bandwidth value available for reservation. Furthermore, engineering metric is assigned to each port by network system administrators [8, 54]. A link provides communication from one router towards another one over two in/out ports in each.

The OSCARS bandwidth reservation system keeps track of changes in the network status and maintains a topology graph which can simply be described as follows. The web service interface enables users to allocate a fixed amount of bandwidth for a time period between two end-points in the network. A reservation request $R$ contains source node $v^s$ and destination node $v^d$, requested bandwidth $\mathtt{M}$, start time $t^s$ and end time $t^e$: $R = (v^s, v^d, \mathtt{M}, t^s, t^e)$. Since there might be bandwidth guaranteed paths in the system that are already fully or partially committed, the reservation engine needs to ensure availability of the requested bandwidth from source to destination for the requested time interval. In order to eliminate over commitment, committed reservations between start and end times are examined to extract available bandwidth information for each link in the time period. The shortest path is calculated based on the engineering metric on each link, and a bandwidth guaranteed path is set up from source to destination, to commit the reservation request for the given time period. Source and destination end-points are usually the host/IP names of the client machines; they are

converted to the corresponding router addresses in the network topology.

**Problem Definition:** Advance network reservation systems like OSCARS enable users to obtain guaranteed requested bandwidth for a certain duration of time. On the other hand, if the requested reservation cannot be granted, no further suggestion is returned back to the user, except a failure message. In such a situation, users have to go through a trial-and-error sequence, and may need to try several advance reservation requests until they get an available reservation. These try-and-error attempts may also overload the system. Even if a user successfully reserves the network, the choice of requested allocation might not be one of the optimal ones available in the system. Further, there is no possibility from the user's point of view to be aware of the other possibilities that might fit better into his/her requirements. In other words, users cannot make an optimal choice. Moreover, the current method of selecting a path may lead to ineffective use of the overall system such that network resources may not be used as optimally as possible.

We enhance the OSCARS reservation system by extending the underlying mechanism to provide a new service in which users submit their constraints and the system suggests possible reservation options satisfying users' requirements.

**Network Reservation Engine:** We developed a new methodology in which users submit constraints and the system suggests possible reservations options. In this approach, instead of giving all reservation details such as the amount of bandwidth to allocate between start/end times, users provide maximum bandwidth they can use, total size of the data requested to be transferred, the earliest start time, and the latest completion time. Moreover, users can set criteria such that they would like to reserve a path for earliest completion time or reserve a path for shortest transfer duration.

Such a request can be represented as: $S = (v^s, v^d, \mathtt{M}^{max}, D, t^E, t^L)$, where $v^s$ nd $v^d$ are source and destination nodes, where $D$ is total size of data to be sent from $v^s$ to $v^d$, and $t^E$ the earliest start time, $t^L$ is the latest end time. The maximum bandwidth $\mathtt{M}^{max}$ is related to the capability of the client and server hosts between source and destination end-points. Even if the network can provide a higher bandwidth than the maximum requested, the user is not be able to use all the available bandwidth

due to limitations and bottlenecks in the client and server sites. The reservation engine finds out a reservation $R = (v^s, v^d, \mathtt{M}, t^s, t^e)$ for the earliest completion or for the shortest duration where $\mathtt{M} \leq \mathtt{M}^{max}$ and $t^E \leq t^s < t^e \leq t^L$.

## 4.2 Methodology and Algorithm

We define the network topology as a time-dependent directed graph $G_T(\mathtt{V}, \mathtt{E}, T)$, with a vertex set $\mathtt{V}$ of $n$ nodes, and an edge set $\mathtt{E} \subseteq \mathtt{V} \times \mathtt{V}$ of $m$ links between nodes. For every edge $k$, $e_k : (v_i, v_j)$, there is a function of available bandwidth $x^{e_k}(t)$ where $t$ is a variable in time domain $T$. The available bandwidth $x^{e_k}(t)$ in $G_T$ is time-dependent, nonnegative, and bounded by an upper limit $u^{e_k}$, where $u^{e_k}$ is the maximum bandwidth available for allocation in $e_k$; such that, $0 \leq x^{e_k}(t) \leq u^{e_k}$ for any instance of time in $T$.

When an advance reservation $R_i = (v_i^s, v_i^d, \mathtt{M}_i, t_i^s, t_i^e)$ is confirmed between start time $t_i^s$ and end time $t_i^e$, we setup a path $\delta_i$ from source node $v_i^s$ to destination node $v_i^d$ that can satisfy the allocation of the requested bandwidth $\mathtt{M}_i$. For every edge along the path $\delta_i : (e_{k^i}, e_{k^j}, \dots)$, we allocate $\mathtt{M}_i$ amount of bandwidth for the future use of reservation $R_i$. The available bandwidth $x^{e_k}$ of each edge in $\delta_i$ is updated in the topology graph $G_T$ for the time period of $[t_i^s, t_i^e]$. It is important to note that $x^{e_k}$ is always $\geq 0$ such that every edge along the path needs to have enough capacity to assure the bandwidth allocation over the path $\delta_i$ setup for reservation $R_i$. Therefore, a reservation request is only confirmed if there exits at least a path from source to destination satisfying the allocation of $\mathtt{M}_i$ bandwidth in the given time period between $t_i^s$ and $t_i^e$.

The example in Figure 4.1 clarifies the underlying mechanism in advance network reservation. At a point of time, assume that there are four reservations confirmed and active in the system; $R_1 = \{A \rightarrow B \rightarrow D, 900Mbps, t_1, t_6\}$, $R_2 = \{A \rightarrow C \rightarrow D, 400Mbps, t_4, t_7\}$, $R_3 = \{A \rightarrow B \rightarrow D, 700Mbps, t_9, t_{12}\}$, $R_4 = \{A \rightarrow C \rightarrow D, 500Mbps, t_9, t_{12}\}$. Thus, the first reservation, $R_1$, is for 900Mbps between $t_1$ and $t_6$ from source $A$ to destination $D$. The system calculated a path based on engineering metric satisfying requested allocation, and allocated bandwidth over $A \rightarrow B \rightarrow D$. $R_2$, $R_3$, and $R_4$ are interpreted

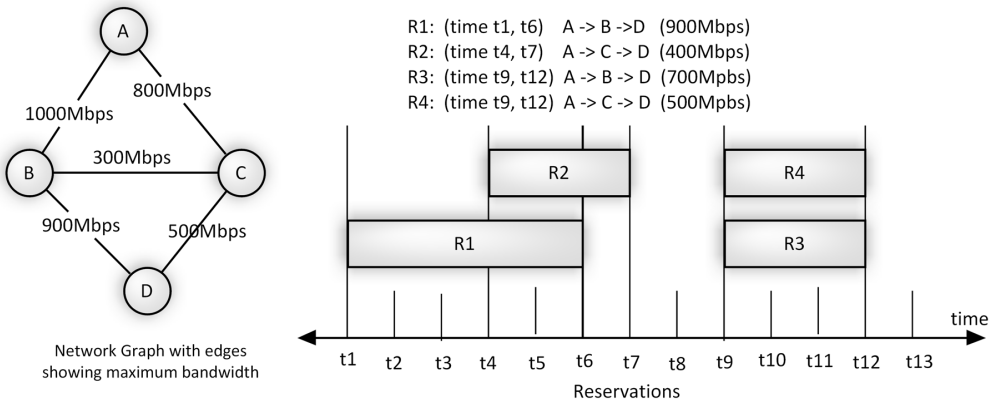



Figure 4.1: Example for Advance Network Reservation

similarly. Figure 4.2 shows the available bandwidth and allocated bandwidth in link $A \rightarrow B$ over time.

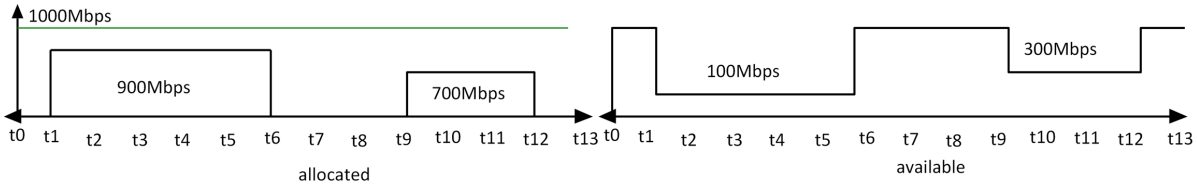



Figure 4.2: Available bandwidth and allocated bandwidth in link $A \rightarrow B$ over time

The first graph in Figure 4.3 represents the status in $[t_1, t_4]$ and the second represents the status in $[t_4, t_6]$. We can confirm a new reservation request from source $A$ to destination $D$ with start time $t_1$ and end time $t_4$, with 500Mbps guaranteed bandwidth, and we can allocate path $A \rightarrow C \rightarrow D$ for the $[t_1, t_4]$ time period. However, we can allocate 100Mbps between $t_4$ and $t_6$. Furthermore, we can only allocate 100Mbps between $t1$ and $t6$ because the maximum amount of bandwidth we can get during the entire period of $[t_1, t_6]$ is 100Mbps. For example, there is an opportunity to send 500Mbps from $A$ to $C$. The maximum flow from $A$ to $C$ is 500Mbps in $[t_4, t_6]$, 100Mbps over $A \rightarrow B \rightarrow C$ and 400Mbps over $A \rightarrow C$. However, we make a reservation for a specific path. Therefore, the maximum amount of bandwidth we can allocate for a single reservation from $A$ to $C$ is 400Mbps in time period $[t_4, t_6]$.

A service request is defined as $S_i = (v_i^s, v_i^d, \mathbb{M}_i^{max}, D_i, t_i^E, t_i^L)$; with total size of data $D_i$ to be sent from $v_i^s$ to $v_i^d$, and and a period of time between earliest start time $t_i^E$ and latest end time $t^L$ such

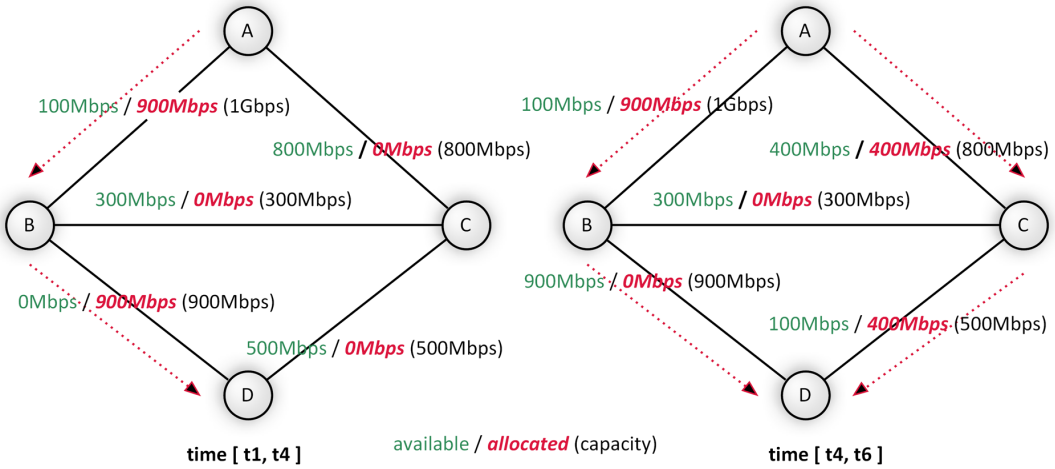



Figure 4.3: Network Flow in specific time periods( $[t_1, t_4]$, $[t_4, t_6]$ )

that, this data transfer need to be accomplished in this given time interval. We search $G_T$ to find possible reservations that can provide enough capacity to transfer the given data volume in the given time period. If there exists bandwidth between $v_i^s$ and $v_i^d$ within the time constraints in $G_T$, a new reservation $R_{earliest}$ for earliest completion time or $R_{shortest}$ for shortest transfer duration is generated. Consequently, we create a reservation $R_j = (v_i^s, v_i^d, \mathtt{M}_j, t_j^s, t_j^e)$ where $\mathtt{M}_j \leq \mathtt{M}_i^{max}$ and $t_i^E \leq t_j^s < t_j^e \leq t_i^L$. We also compute a path $\delta_j$ satisfying reservation $R_j$.

In order to satisfy the given criteria, the amount of bandwidth allocation $\mathtt{M}_j$ and the time interval $[t_j^s, t_j^e]$ need to be sufficient to transmit data volume of $D_i$ using the path $\delta_j$ allocated for reservation $R_j$. We can simple say $D_i = \mathtt{M}_j \times d$ where $d$ is the duration between start time $t_j^s$ and end time $t_j^e$. $R_{shortest}$ has the minimum duration $d = |t^s, t^e|$ among all other possible reservation satisfying $S_i$. The objective for earliest completion time is to select a reservation $R_j$ satisfying the criteria given in $S_i$ which has the earliest end time $t^e$. On the other hand, we would favor a reservation with a shorter duration if there are more than one possible reservations completing at the same earliest time. For reservation $R_{earliest}$, $\forall R_j$ satisfying $S_i$: $t^e_{earliest} \leq t^e_j$, and $\forall R_j$ with $t^e_j = t^e_{earliest} : t^s_{earliest} \geq t^s_j$.

## 4.3 Search Interval between Earliest-Start and Latest-End Times

The outline of our approach is as follows. We divide the given search interval into time steps. The search interval $[t_i^E, t_i^L]$ is the time period between earliest start time $t_i^E$ and latest end time $t_i^L$ in which the data need to be transmitted. A time step represents the longest duration of time in which we have a stable discrete status in terms of available bandwidth over the links. A time period $[t_i, t_j]$ is considered as a time step if $\forall e_k \in G_T : x^{e_k}(t) = c_k$ where $t_i \leq t \leq t_j$, and $c_k$ is a constant. We obtain a static directed graph that keeps information about the available bandwidth status for every link. This information is updated on-the-fly every time a reservation request is committed and stored for further processing during the path calculation phase. A snapshot graph of $G_T$ in time step $ts(t_i, t_j)$ is defined as $G(ts_i)$, with the same vertex set and same edge set. For every edge $e_k : (v_i, v_j)$ in $ts(t_i, t_j)$, the available bandwidth $x^{e_k} = c_k$ stands for the value of $x^{e_k}(t)$ in $G_T$ between $t_i$ and $t_j$ in time step $ts(t_i, t_j)$. This helps us discretize the dynamic graph and apply known graph algorithms efficiently.

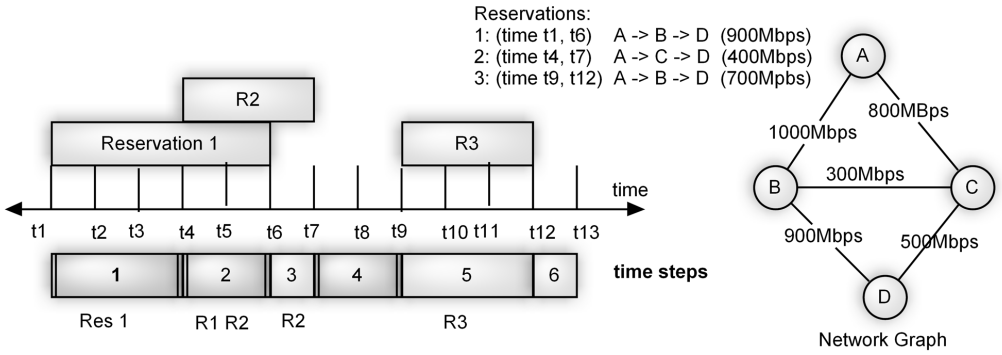



Figure 4.4: Time Steps

Figure 4.5 shows time steps between $t_1$ and $t_{13}$, for the example given in Figure 4.1 with four committed reservations. We have six time steps: $ts_1(t_1, t_4)$, $ts_2(t_4, t_6)$, $ts_3(t_6, t_7)$, $ts_4(t_7, t_9)$, $ts_5(t_9, t_12)$, $ts_6(t_12, t_13)$. Every time step corresponds to a static snapshot of the network topology. Figure 4.6 shows $G(ts_1)$, $G(ts_2)$, $G(ts_3)$, $G(ts_4)$, $G(ts_5)$, and $G(ts_6)$.

We analyze the search interval $[t^E, t^L]$ with a set of consecutive time steps covering the entire period; $\{t_{S\,k}(t_i, t_{i+1}), t_{S\,k+1}(t_{i+1}, t_{i+2}), \ldots, t_{S\,k+n}(t_{i+n-1}, t_{i+n})\}$, where $t_i < t_{i+1} < t_{i+2} \cdots < t_{i+n}$, and $t^E \geq t_i$ and $t^L \leq t_{i+n}$ The set of confirmed reservations in the system characterize time steps since they change

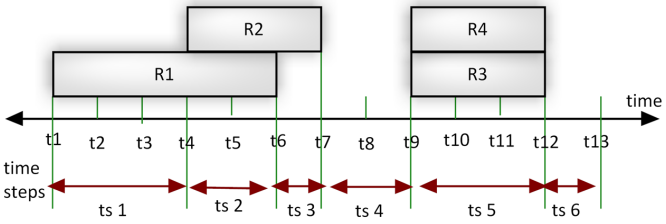



Figure 4.5: Time steps between $t_1$ and $t_{13}$

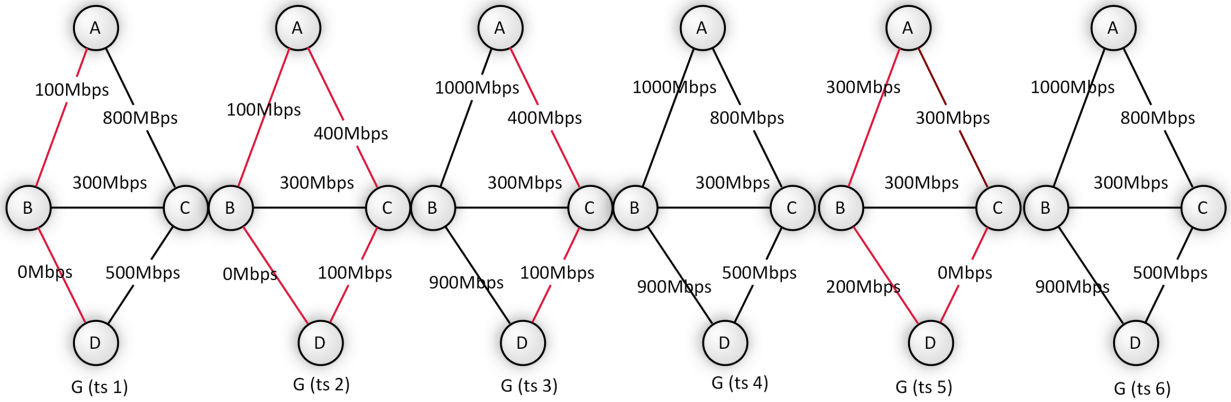



Figure 4.6: Static Graphs for time steps $ts_1$, $ts_2$, $ts_3$, $ts_4$, $ts_5$, $ts_6$

the available bandwidth values in the network topology. If two reservations partially overlap in terms of time period, they split the total period of time into either two or three time steps. If they do not overlap, they split into three time steps. In other words, the number of time steps in the search interval is bounded by the number of committed reservations within the given period $[t^E, t^L]$. If there are $r$ committed reservations falling into the period, there can be maximum $2r + 1$ different time steps in the worst-case. Figure 4.5 shows the general idea behind time steps and reservations.

The next step is to traverse these time steps to check whether we can find a reservation satisfying the given criteria. For the example given in Figure 4.5 and Figure 4.6, first $ts_1$, and then $ts_2$ will be examined; later, if both cannot satisfy the request, time window $tw(t_1, t_6)$, a combination of $ts_1$ and $ts_2$, will be examined. A time window consists of subsequent time steps. $tw_k$ is a time window which corresponds to the time period in $ts_k$. $tw_{k_1-k_2}$ is a time window including all time steps between $ts_{k_1}$ and $ts_{k_2}$. If there are $s$ time steps in a given search interval, there are $(s \times (s + 1))/2$ time windows since time windows are subsequent combinations of time steps.

We search through these time windows in a sequential order to check whether we can satisfy the requested allocation in that time window. For a bandwidth allocation with the shortest duration, we can sort time windows according to their length, and start checking with the smallest one. For a bandwidth allocation with the earliest completion time, we can benefit from a specific search pattern. The search pattern for earliest completion time in the given example will be as follows: $tw_1, tw_2, tw_{1-2}, tw_3, tw_{2-3}, tw_{1-3}, tw_4, tw_{3-4}, tw_{2-4}, tw_{3-4}, \ldots$. The algorithm will stop searching when it finds a time window satisfying the given criteria. In most cases, we do not need to check all possible time windows. In the worst-case, we may require to search all time windows, which makes $(s \times (s+1))/2$ searches, where $s$ is the number of time steps.

## 4.4 Examining Time Windows to Find Possible Reservations

While checking a time window to verify whether it can satisfy the request, we first look at the total duration of the time window. We know the max bandwidth $\mathtt{M}^{max}$ user can support, and the total size of data $D$. Therefore, we first determine the duration of a time window and simply ensure whether this time window is large enough to satisfy the user request. The length of a time window $d = |tw_{k1-k2}|$ should be larger than the minimum amount of time, $D/\mathtt{M}^{max}$, required to transmit data if $\mathtt{M}^{max}$ bandwidth can be allocated.

Then, we calculate the maximum bandwidth available from source $v^s$ to destination $v_d$ in time window $tw$. We use max-bandwidth path algorithm over static snapshot graph $G(tw)$. $G(tw)$ can easily be computed using snapshots of time steps that form this time window. $G(tw_k) = G(ts_k)$, and $G(tw_{k_1-k_2}) = G(ts_{k_1}) \circ G(ts_{k_1+1}) \circ G(ts_{k_1+2}) \cdots \circ G(ts_{k_2})$. We define a new operator, $\circ$, to intersect static snapshot graphs. $G_1 \circ G_2$ forms a new graph with the same vertex and edge set as in $G_1$ and $G_2$. For each edge $e_k$, the available bandwidth is the minimum of $x^{e_k}$ both in $G_1$ and $G_2$. Such that, $\forall e_k \in G_1 \circ G_2 : x^{e_k} = min\{x_1^{e_k}, x_2^{e_k}\}$, where $x_1^{e_k}$ is the available bandwidth of $e_k$ in $G_1$ and $x_2^{e_k}$ is the available bandwidth of $e_k$ in $G_2$. This property makes the process easy, since we only need to store one graph snapshot for each starting time window; for example, to obtain $G(tw_{1-3})$, we only need

$G(tw_{1-2})$ and $G(tw_3)$, $G(tw_{1-3}) = G(tw_{1-2}) \circ G(tw_3)$.

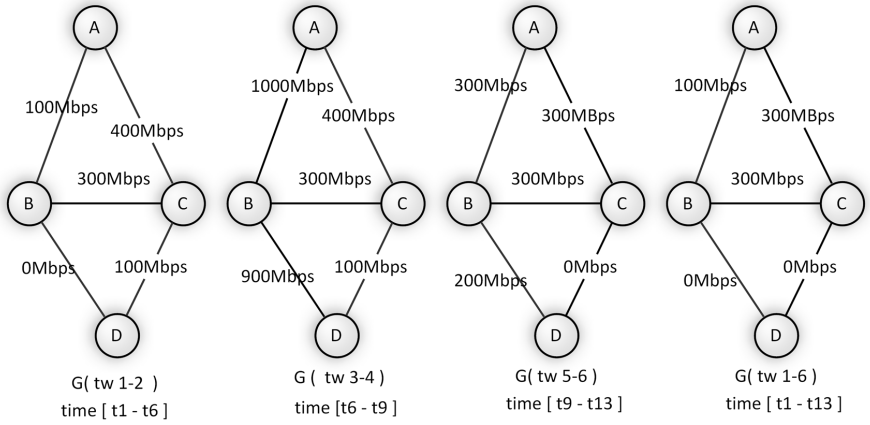



Figure 4.7: Static Graphs for time windows $tw_{1-2}$, $tw_{3-4}$, $tw_{5-6}$, and $tw_{1-6}$

Figure 4.7 shows static snapshot graphs for time windows $tw_{1-2}$, $tw_{3-4}$, $tw_{5-6}$, and $tw_{1-6}$. $G(tw_{1-2}) = G(ts_1) \circ G(ts_2)$, $G(tw_{3-4}) = G(ts_3) \circ G(ts_4)$, $G(tw_{5-6}) = G(ts_5) \circ G(ts_6)$, and $G(tw_{1-6}) = G(tw_{1-2}) \circ G(tw_{3-4}) \circ G(tw_{5-6})$. $R_1$ and $R_2$ are active in time interval $[t_1, t_6]$, so links associated with both $R_1$ and $R_2$ are updated in $G(tw_{1-2})$. Only $R_2$ is active in time interval $[t_6, t_9]$, so links associated with $R_2$ are updated in $G(tw_{3-4})$.

While exploring a time window, a max-bandwidth path $\delta$ is calculated in $G(tw)$ in which $\mu_{tw}(v^s, v^d)$ is the maximum amount of bandwidth we can allocate in time window $tw$. $d_{tw} \times \mu_{tw}$ simply gives the amount of data that can be transmitted if a reservation is made in time window $tw$, where $d_{tw}$ is the length of the time window. A time window $tw(t_i, t_j)$ is selected and marked if it can provide enough resources to satisfy the user criteria. For such a time window, $d_{tw} = |max\{t_i, t^E\}, min\{t_j, t^L\}|$ is the maximum duration we can use to make a reservation, and $\mu_{tw} = \mu_{tw}(v^s, v^d)$ is the maximum amount of bandwidth we can allocate from source to destination. Note that we need to the consider amount of bandwidth we can use which is also limited by the maximum set by the user, $\mu'_{tw} = min\{\mu_{tw}, \mathtt{M}^{max}\}$. Therefore, the product $\mu'_{tw} \times d_{tw}$ should be greater than the requested volume size $D$.

When a satisfactory window is found, we generate a reservation $R = (v^s, v^d, \mathtt{M}, t^s, t^e)$ and a path from source to destination to be used for this reservation in the network. The start/end times and $\mathtt{M}$ are calculated based on the given user criteria and available resources in the time window. A

straightforward strategy to generate a reservation when a time window $tw$ is selected and marked to satisfy the user criteria is as follow: $t^s = max\{t_i, t^E\}$, $\mathtt{M} = min\{\mu_{tw}, \mathtt{M}^{max}\}$, and $t^e = t^s + |D/\mathtt{M}|$.

**Input**: A set of time steps in the search interval $\{ts_1, ts_2, \ldots, ts_n\}$
**Output**: A network reservation for earliest completion or shortest duration
**for** $i = 1$ *to* $n$ **do**
  **for** $j = i$ *to* 1 **do**
    Get time window $tw = tw_{j-i}$ which contains all time steps between $ts_j$ and $ts_i$;
    **if** *the given criteria can fit into the time window* $tw = ts_j \ldots ts_i$ **then**
      Obtain static snapshot graph $G(tw)$ for time window $tw$;
      Calculate max-bandwidth $\mu_{tw}$ from source to destination;
      **if** *we can satisfy request in time window tw (Examine* $\mu_{tw}$*)* **then**
        select $tw$ ;
    **if** *goal is to find a reservation with Earliest completion* **then**
      **if** *there is any selected time window tw* **then**
        Get $tw$ with shortest duration to satisfy the given request;
        Generate a Reservation and a Path, Return for earliest completion;
**if** *goal is to find a reservation with Shortest duration* **then**
  **if** *there is any selected time window tw* **then**
    Get $tw$ with shortest duration to satisfy the given request;
    Generate a Reservation and a Path, Return for shortest duration;
Return: No reservation found;

**Algorithm:** A sample search pattern to find a reservation with earliest completion time or shortest transfer duration

Figure 4.8 shows the search pattern to find a reservation for the earliest completion time, for the example given in Figure 4.1. Assume that we have a service request $S = (A, D, 200Mbps, 200 \times 4t, t_1, t_13)$, and we want to find a reservation satisfying the given criteria. Time window $tw(t_1, t_4)$ with length $3t$, and time window $tw(t_4, t_6)$ with length $2t$, are short in duration to conform to the requirements of this request. The maximum bandwidth allowed is 200Mbps, so we need at least a time window with length $4t$. $tw(t_1, t_6)$ satisfies the time requirement, so we proceed and calculate the maximum bandwidth available in $G(tw(t_1, t_6))$. The maximum bandwidth we can reserve from $A$ to $D$ between $t_1$ and $t_6$ is 100Mbps. Total size of data we can transfer is $100 \times 5t$. Therefore, $tw(t_1, t_6)$ can not satisfy the bandwidth requirement. We keep searching through time windows until we find $tw(t_1, t_9)$ which satisfies both time and bandwidth requirements. Time window $tw(t_1, t_9)$ is selected for the earliest completion time. We generate $R_{earliest} = (A, D, 100Mbps, t_1, t_9)$ with start time $t_1$ and end time $t_9$. If we want to find a reservation for the shortest transfer duration, we need to continue

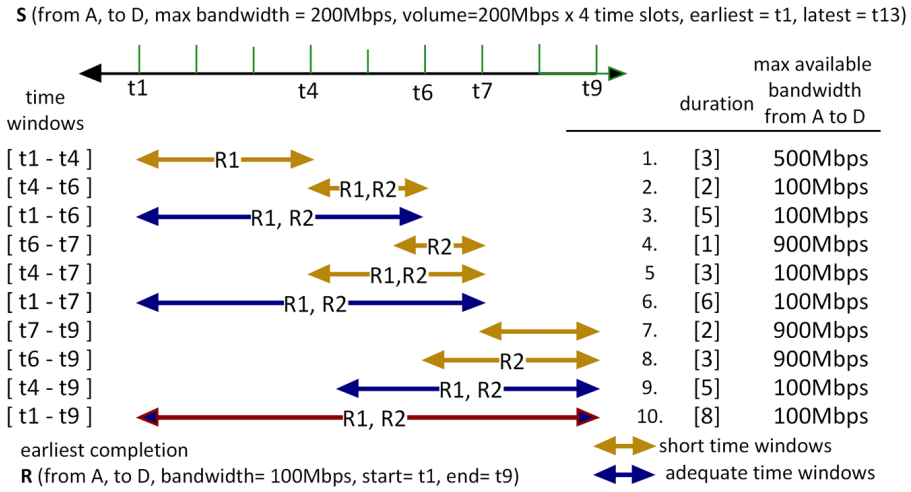



Figure 4.8: Example for earliest completion

searching until we cover the entire interval between $t_1$ and $t_{13}$. As shown in Figure 4.9, $tw(t_9, t_{12})$ and $tw(t_7, t_{12})$, $tw(t_6, t_{12})$, $tw(t_4, t_{12})$, $tw(t_1, t_{12})$, $tw(t_{12}, t_{13})$, $tw(t_9, t_{13})$ ...are searched next. Time window $tw(t_9, t_{13})$ satisfies the given bandwidth and time requirements. All other time windows coming after this in the search pattern, are longer in terms of duration. Therefore, $tw(t_9, t_{13})$ gives the reservation $R_{shortest} = (A, D, 200Mbps, t_9, t_{13})$ with shortest duration. If the total volume of data is $175 \times 4t$, then the search will be same with $R_{shortest} = (A, D, 200Mbps, t_9, t_{12.5})$ and $R_{earliest} = (A, D, 100Mbps, t_1, t_8)$.

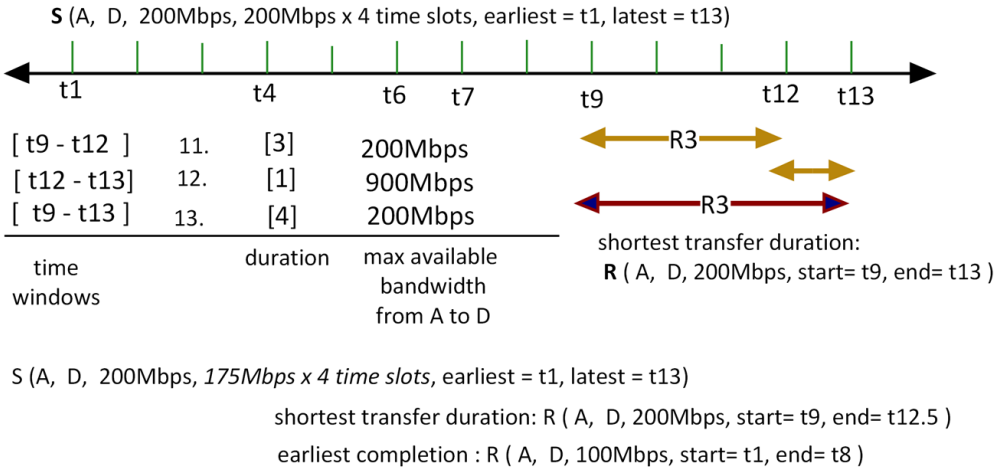



Figure 4.9: Example for shortest transfer duration

## 4.5 Evaluation of the Proposed Algorithm

Max bandwidth path algorithm is bounded by $O(n^2)$, where $n$ is the number of nodes in the topology graph. In the worst-case, we may require to search all time windows, $(s \times (s + 1))/2$, where $s$ is the number of time steps. If there are $r$ committed reservations in that period, there can be a of maximum $2r + 1$ different time windows in the worst-case. Overall, the worst-case complexity is bounded by $O(r^2n^2)$. However, $r$ is relatively very small compared to the number of nodes $n$, in the topology. Bandwidth reservation is used for large-scale data transfers and it is very unlikely to have thousands of committed reservations in a given time period. Also, the path calculation from two end-points does not span to all nodes in a real network; therefore, we can trim the topology graph and perform calculation on a reduced data set while calculating path from source to destination. Moreover, time windows that are too short in duration to transmit the requested amount of data are eliminated beforehand. Max bandwidth and shortest path algorithms are quite efficient and the search process over time windows is scalable and practical, considering that the number of reservations in practice is limited. Furthermore, there are usually less than a hundred node in a typical network topology like ESnet. We have tested the performance of the algorithm by simulating very large graphs (with 10K nodes) and we have observed that the computation time is in the order of seconds.

# Chapter 5

# Scheduling with Time and Resource Conflicts

In this chapter, we analyze scheduling of data transfer operations with time and resource constraints. We explain several common approaches used in the literature to analyze file scheduling and resource assignment, and we emphasize the complexity of the scheduling problem with resource and time conflicts. We give several cases to clarify the problem domain by highlighting methodologies and special techniques used to obtain a solution. Our main concentration is arrangement of data transfer operations in time dependent networks. The bandwidth assignment, which has been described in previous chapters as well, plays an important role designing a solution. In our scheduling paradigm given in this study, the transfer rate is fixed and does not vary over time. This is one of the crucial features that affect the methodologies used to approach the problem. In order to elucidate the problem domain and introduce the concepts, we present the crucial decision points in the process of designing a scheduling algorithm with resource and time constraints. We present several recent studies in the literate solving similar but simpler use cases. We also compare offline greedy algorithms with online algorithms in which no prior knowledge about the future states is available. We conclude this chapter with a simple example to make readers more familiar with the theory behind scheduling with time and resource constraints with fixed bandwidth assignment.

## 5.1 File Transfer Scheduling in a Network

There are several studies in the literature [37] categorizing several research problems in data transfer scheduling [43, 97], and summarizing theoretical complexity and difficulty of those problems in several domains [57]. A simple scheduling problem can be defined as follows.

We have a network consisting of $n$ nodes connected to each other, and $m$ files to be transferred over this network. Each node can transfer up $c_i$ concurrent files at a time, and a file need to be sent

from node $i$ to node $j$. Each file may have different data size; therefore, the total amount of time to complete the transfer may vary for each file since it directly depends on the size of data sent over the connection between two nodes. One common objective is to find a schedule in which we minimize the total amount of transfer time. We organize each file transfer in such a way that the makespan is minimized. This is a general type of assignment problem and it is NP-complete [57].

On the other hand, the structure of the network has direct relation with the complexity of the problem. [57] analyzes some common cases and shows that there are polynomial time solutions for some very special types of the problem. Other than that, the general problem is proven to be NP-hard. The study given in [57] examines several network structures such as trees, bipartite graphs, networks with odd and even cycles, and provides a detailed complexity analyze through relaxing the problem by eliminating parameters such as file size and concurrency.

A very straightforward file transfer scheduling problem can be defined if we eliminate the concurrency and file size parameters. We let nodes transfer only one file at a time, $c = 1$; and files have same data sizes, so the duration of file transfers are same. Such a simplified version can be a cardinality matching problem. It is a well known that analyzing bipartite graphs has many benefits. If the network is bipartite graph, sender and receiver nodes are separated from each other; we can simplify it as a bipartite cardinality matching problem. Bipartite cardinality matching can easily be solved by converting it to a max-flow problem. We would like to emphasize that we evaluate a specific network type; beyond that, important parameters in the problem, file size and concurrency, have no effect and not included in this simple case.

We call attention to the definition of the simple problem given above. We have no time dependency and there is no capacity in nodes, also between nodes. All transfer jobs, files, are given at the beginning and we try to find an offline scheduling to minimize the total transfer time. Some special cases can be solved by generalizing to graph coloring and bipartite matching problems. Further, the broader problem is quite hard and needs much attention. We would like readers note that we are dealing with assignment problem which are typically NP-complete. The scheduling with advance

reservation brings time dependency for each job and a dynamic network infrastructure as well. Furthermore, we study a practical online algorithm which makes decisions on the fly when new requests arrive.

## 5.2 Resource Assignment

We can apply a more common technique using bipartite graph structures, where source and destination end-points are separated, to model a special case of the problem. The Hungarian algorithm [51] is one of the well known combinatorial algorithms that gives a polynomial time solution to the assignment problem. There are $m$ tasks and $n$ nodes, and each task has a specific cost/profit when assigned to a particular node. Given that we know the cost/profit matrix, the objective is to maximize the profit or minimize the cost by finding an optimum mapping of tasks. For this case, tasks represent the file transfer operations; and, we would need to find an optimum mapping of file transfer operation into nodes pairs. Let assume that we still transfer single file at a time, we have concurrency limitations, which is one. On the other hand, transfer rates between nodes, bandwidth, is not unique in this case. We extend the simple problem by adding another parameter; such that, we also consider the transfer rate between node pairs. We can model the profit function by file sizes. We want to maximize the total amount of data transferred at a certain amount of time. We can benefit from the Hungarian method in order to model such a system in which we select a set of files in order to maximize the total throughput given that each edge has a particular capacity.

If we allow concurrent transfers, the problem cannot be represented using the same model. The assignment problem becomes more complex, and the solution becomes harder, NP-hard. Concurrent transfers enable total bandwidth to be shared between multiple operations. However, in real life we usually do not apply a concurrency limit; instead, we have maximum capacity in resources which confines the maximum number of transfer operations at a time. In other words, we may start sending many files at a time which have small transmit rates. If we allow bandwidth sharing and try to find an optimum for how resources are split among operations, we end-up in a more difficult dilemma.

Designing an online scheduling algorithm has difficulties compared to offline scheduling in which we have all job information in advance [56]. A common methodology is to examine arriving jobs at a period of time. The periodic scheduling brings some advantages by delaying the decision time and making a schedule by granting a set of recent jobs together.

## 5.3 Time Constraint in Fixed Bandwidth Assignment

File transfer scheduling with a specific start time and a particular deadline has been studied in literature [74, 94, 84]. The scheduling problem has been formulated as a multi-commodity flow problem, and uniform time slices have been used to model the time dependency in [84]. The objective is to maximize the total transfer throughput and data transfers can use varying bandwidth in every time slice. This problem can be generalized as a concurrent file transfer problem [94]; such that, we share the bandwidth between multiple jobs and try to utilize the network as much as possible. Using network flows to model and place a solution space to combinatorial optimization problems is a common practice [74]. On the other hand, sharing bandwidth between concurrent transfers can improve the total throughput but does not help satisfying completion time of each job. Our objective is to provide allocation of scheduling time not to improve the system utilization.

We would like to emphasize that multi-commodity flow does not apply our case. Recall the analogous example, time steps and time windows, given in previous chapters. There are several practical ways to set the transfer rate. One of them is to adjust the number of parallel streams which will be discussed in dynamic adaptation in the following chapters. Furthermore, there are several approaches such as modifying the TCP stack in kernel to limit and control the transfer rate. Moreover, we have already studied network reservation and guaranteed bandwidth allocation in which the network itself limits the total usage of bandwidth. Although it is has many practical issues in real life applications, solving fixed bandwidth problem is much harder. Therefore, we explain the unsplittable flow problem to help model the fixed bandwidth assignment problem.

The unsplittable flow problem [62] is an interesting dilemma in algorithm research. We can simply

describe it by $t$ tasks with start and end time and a particular demand $d > 0$ and a profit $p$. If we assign a task, it requires $b_i$ amount of bandwidth. We are given a network with available bandwidth $b_t$ for every time $t$. The purpose is to find a subset of tasks to maximize the profit. Similarly if every task acquires a cost value, objective is to minimize the cost. The unsplittable flow problem is NP-hard, and only polynomial approximation algorithms are given in [17, 26, 61, 59, 35]. Interestingly, even for very special cases (i.e. planar graphs) the problem is still NP-hard.

In order to clarify the concept behind fixed bandwidth time dependent scheduling in a distributed network, consider a single network with a single line. We let only one edge connecting two nodes. In this special case of the general problem where network is a line, the unsplittable flow problem converts to a very well known optimization problem, Knapsack problem [75]. In Knapsack problem, we have a set of items each with a weight and a cost value, and we select a collection of items to maximize the total profit considering that we have a limit in total capacity. Similarly, we have start and end times for each task, and we have the bandwidth limit over the link. Beyond that, even if we have unique profit $p = 1$, and unique demand $d = 1$ for all tasks, and we set the edge capacities to a unique value, we still end up with a NP-hard problem. This special case can be generalized to maximum edge-disjoint paths problem in graph theory [26].

In the following section, we elaborate on the methodology used to explore time and resource constraints and evaluate conflicts. We study unsplittable flow problem and give a model to clarify the problem domain. We state an example to show the complexity of scheduling in dynamic network with time constraints. The problem we attempt to solve is quite hard. To the best of our knowledge, only polynomial approximation algorithms have been proposed in literate as discussed above, and there is no constant factor approximation algorithm known to solve the unsplittable flow problem.

One way to cope with hard problems is to design approximation algorithms in which we set priorities and rate each selection to reduce the search space. As we describe in the following section, the number of possible options to examine in order to make the best selection exponentially increases in worst case. Instead of that, we rate each selection and displacement based on the priority or the

cost/desire we assign to each task. Thus, we can design polynomial greedy heuristics which can solve the problem with a near optimum scheduling choice. Note that very simple but effective greedy approaches like best-fit, first-come, and earliest-deadline, use some preference/criteria to make a choice among multiple options. The design of the algorithm and deciding on a good selection criteria play important role in terms of the quality of the resulting scheduling approximation. There are many studies in the literature investigation approximation algorithms for scheduling; [56] and [44] are one of them which show benefits in designing greedy algorithms with priorities .

## 5.4 Analyzing the Assignment Problem in a Distributed Network

We define a sample network with three data transfer nodes connected to each other over a network. Each node has a particular capacity that it can provide maximum upload and download transfer rate. This defines the limit in server site such that total throughput is also constrained by the capacity of data transfer nodes. Figure 5.1 gives a sample example with three jobs. Each job has an amount of data need to be transferred, and a specific period of time this job need to be completed - earliest start time $t^E$, and latest completion time $t^L$.

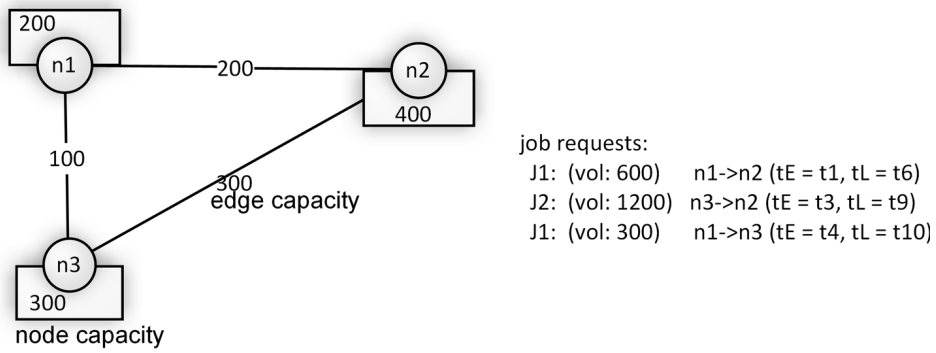



Figure 5.1: Sample Problem Definition

As we have described in the previous sections, we are bounded by edge capacity as well as node capacities. Figure 5.2 shows the resource conflicts in this simple example. If we have a transfer request from node $n_1$ to $n_2$ running at the same with another request from node $n_3$ to $n_2$, the total bandwidth allocation given to both should not exceed the capacity of the shared node $n_2$.

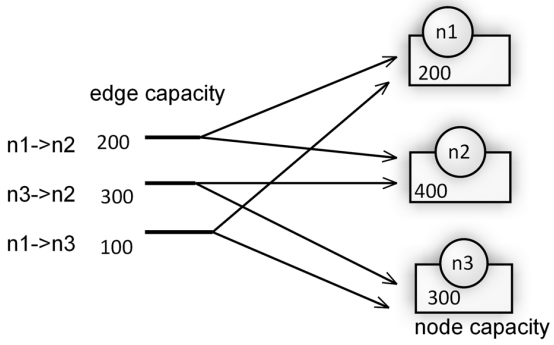



Figure 5.2: Node and Link - Resource Contraints

Earliest start time and latest completion time of each job defines its search interval. We focus on the search interval to find a proper allocation for the given request. Figure 5.3 shows time steps which are calculated according to constraints of each job. A time step shows the longest duration of time in which we have a stable network structure in terms of available bandwidth ready for reservation. Figure 5.4 shows time windows. We traverse time windows in a specific order. A time window is a sequence of time steps. First we try to find an allocation which has shortest duration of time; or simply say which includes less time steps. Besides, we want to find an allocation with earliest completion; so, we traverse first time windows which end earlier. Time steps and time windows have been described in details in previous chapters. We further demonstrate mapping time windows to search intervals of each job in Figure 5.4.

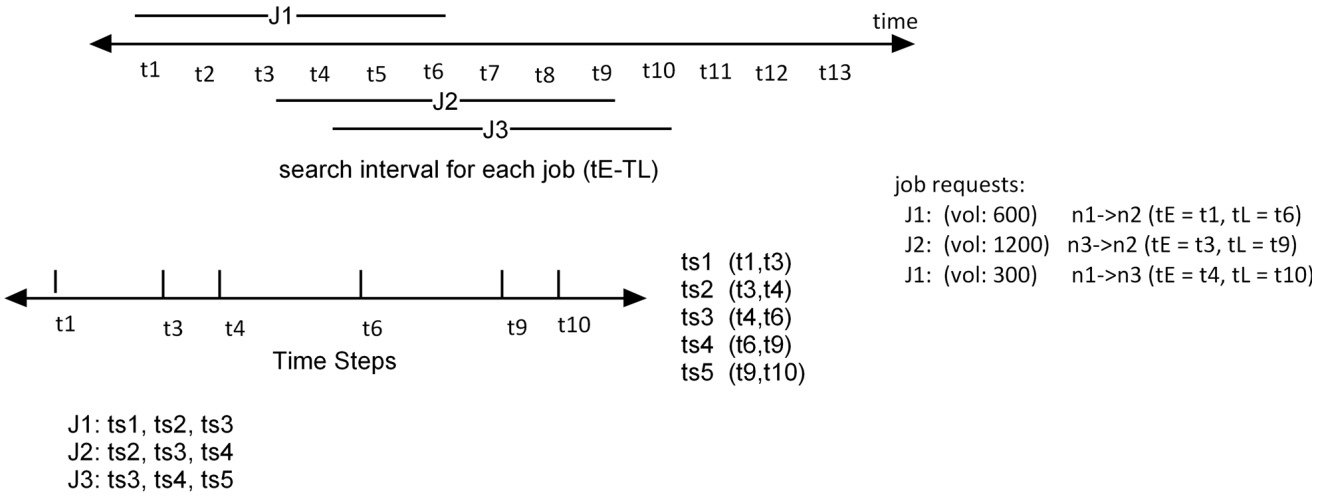



Figure 5.3: Time Steps and Search Interval

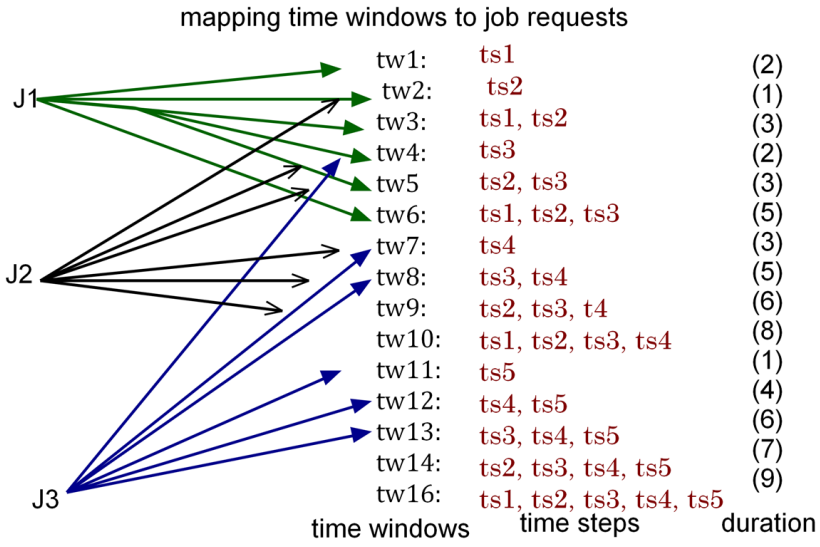



Figure 5.4: Time Windows

The total amount of data for each job that need to be transferred characterizes the duration of the time period needed. Figure 5.5 shows how several time windows are eliminated in the search interval. Further, it also illustrates the resource constraints specific to each time windows. For example, if we want to assign job $J_1$ into time window $tw_6$, we need at least a capacity of 120 allocated over the link from $n_1$ to $n_2$ which provides maximum of 200 capacity. However, we would first consider $tw_3$ and $tw_5$ if there is more capacity available since those time windows consist of less time steps (shorter duration). Figure 5.6 provides a more detailed view of the assignment options.

We have analyzed the unsplittable flow problem. If we could solve that in a polynomial time, we would also solve this problem. Figure 5.7 represents the sample problem using network flows. The crucial point is that each time window may affect more than a single time step. And, those time steps need to have the same capacity allocation during the entire period of time in this time step. As an example, $tw_9$ and $tw_5$ both include $ts_2$. Any flow passing over these two should also consume capacity in $ts_2$. Even though we could represent the network structure with discrete graphs in each time step, we still need to consider time constraints. In other words, our problem complexity increases exponentially when we have time constraints and resource constraints together. In this case, we have resource conflicts in each time step, see Figure 5.2, and time conflicts for time windows as shown in Figure 5.7.

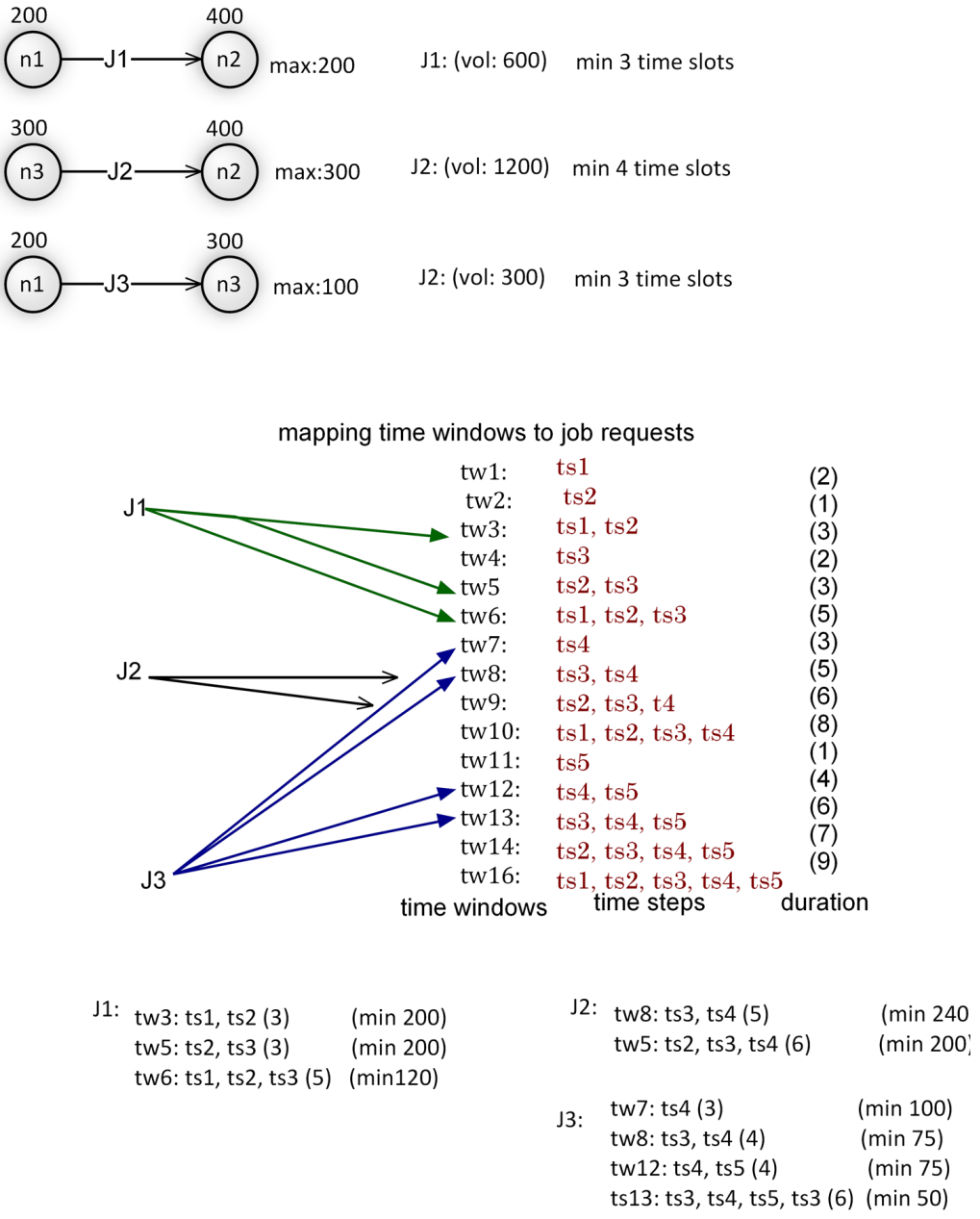



Figure 5.5: Mapping Jobs to Time Windows

Figure 5.8 provides a table of possible assignment options that need to be considered with resource constraints given in Figure 5.9. In Figure 5.10, we present how solution space is analyzed in this sample problem. We show sample conflicts, and explain that search space is exponential. We have three possible assignment option for $J_1$, two for $J_2$, and four for $J_3$. Overall, we may need to consider $3 \times 2 \times 4$ choices in order to make a selection. Each assignment might affect other options, but there is no direct correlation between them.

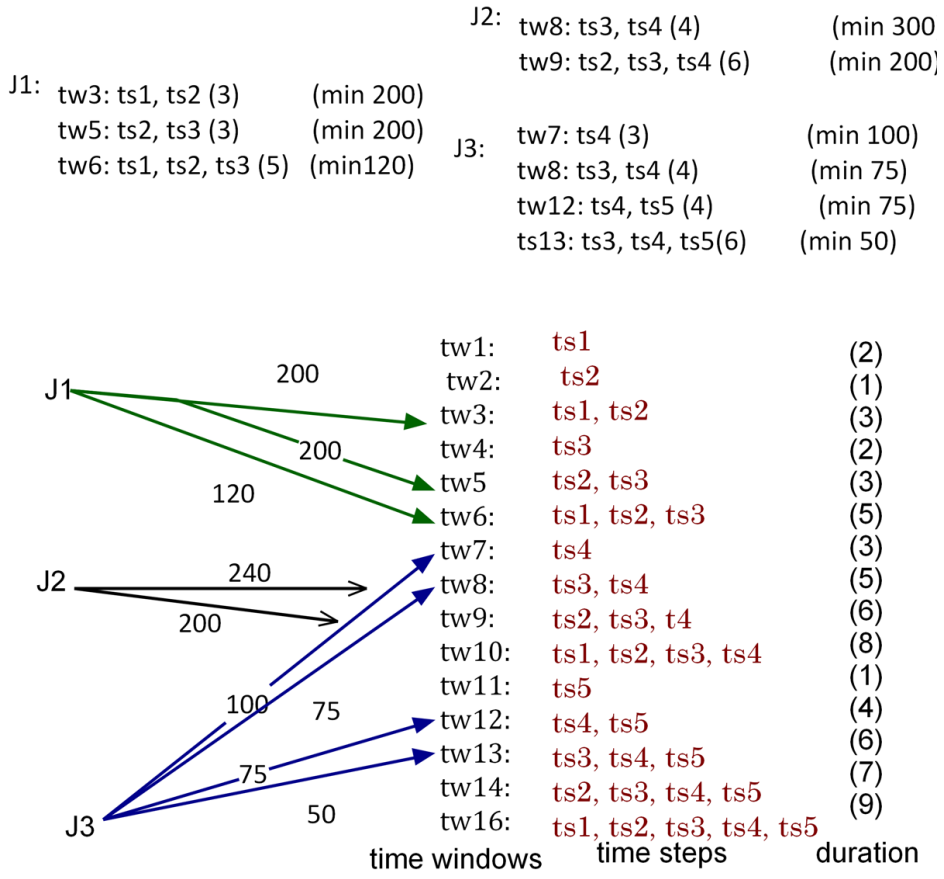



Figure 5.6: Minimum Capacity required for each Time Window

For example, if we select $tw_8$ for $J_2$, we could assign $J_3$ into time window $tw_{13}$. $tw_8$ includes $ts_3$ and $ts_4$, a period of time between $t_4$ and $t_9$. The minimum capacity we can use in this time window for job $J_2$ is 240, but we can finish by $t_8$ if use 300. In such a case, we would not be able to assign $J_3$ into $tw_{13}$ since there we be no capacity left at node $n_3$. However, if total volume of job $J_3$ was 200 instead of 300, we could assign it between $t_8$ and $t_{10}$. Assume that total volume of data for job $J_3$ is 200. The flow from $J_3$ to $tw_8$ and $tw_{12}$ would be 50 instead of 75. In such a case, we would be able to select $tw_8$ for $J_2$, and $tw_{12}$ for $J_2$ (total makes $240 + 50 = 290 < 300$). We could use 240 amount of bandwidth for $J_3$, and 50 amount of bandwidth for $J_3$, between $t_4$ and $t_9$. Alternatively, we could use 300 amount of bandwidth for $J_3$ between $t_4$ and $t_8$, and 100 amount of bandwidth for $J_3$ between $t_8$ and $t_{10}$. In other words, we would introduce a new point at $t_8$ and divide the time step $ts_4$ into two. Even though unsplittable flow model given in this sample example is capable to find an assignment options, we would need further processing to purify results in order to obtain the best selection.

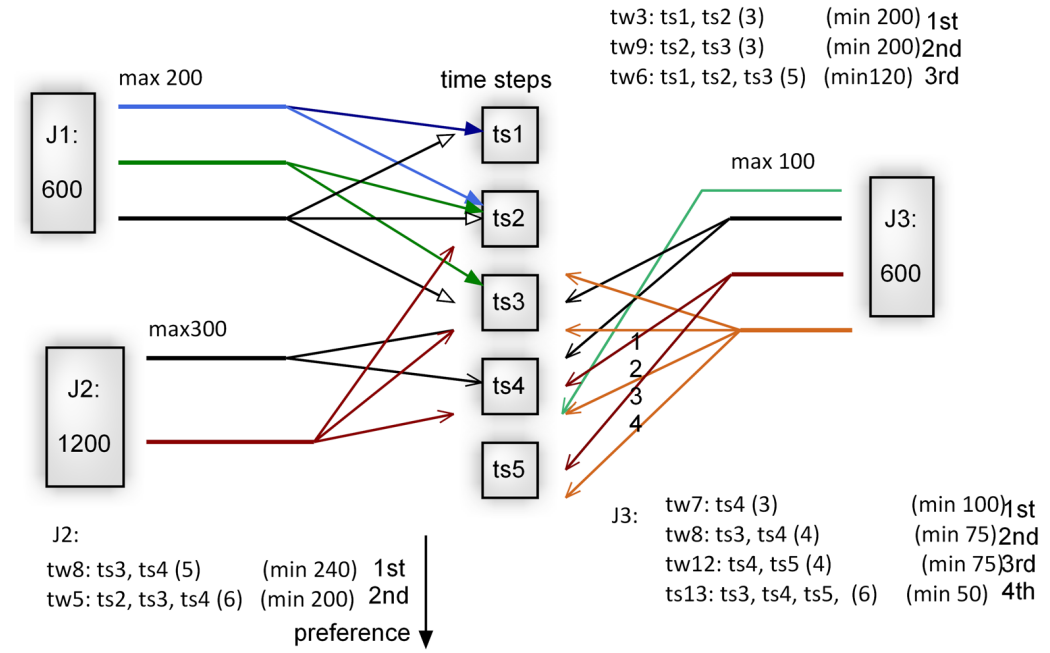



Figure 5.7: Time Conflicts

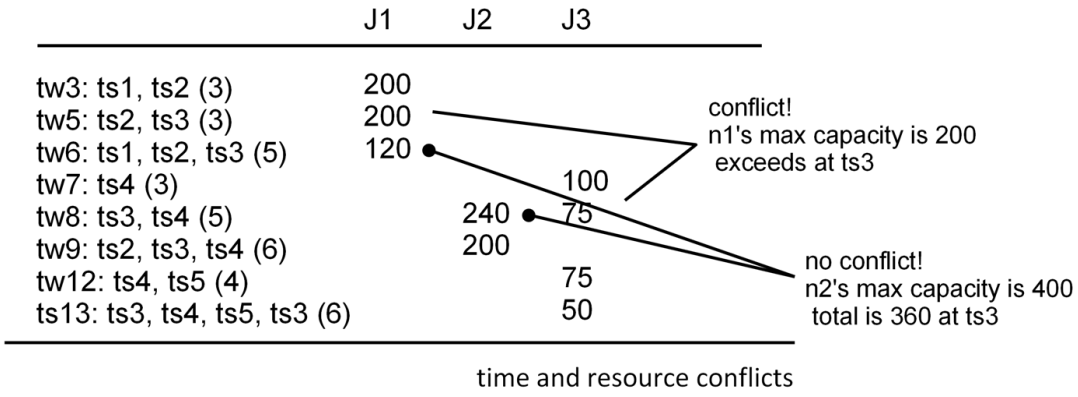



Figure 5.8: Time and Resource Conflicts

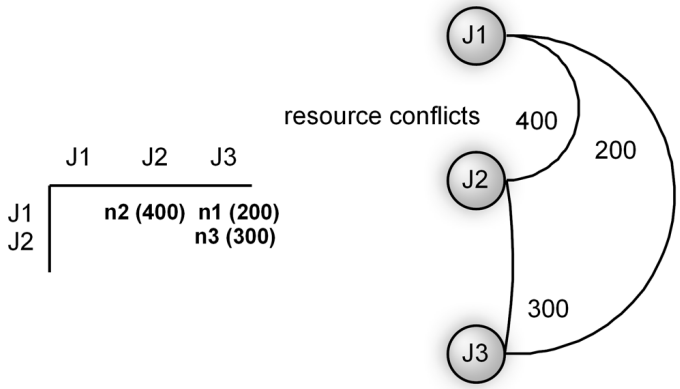



Figure 5.9: Resource Conflicts

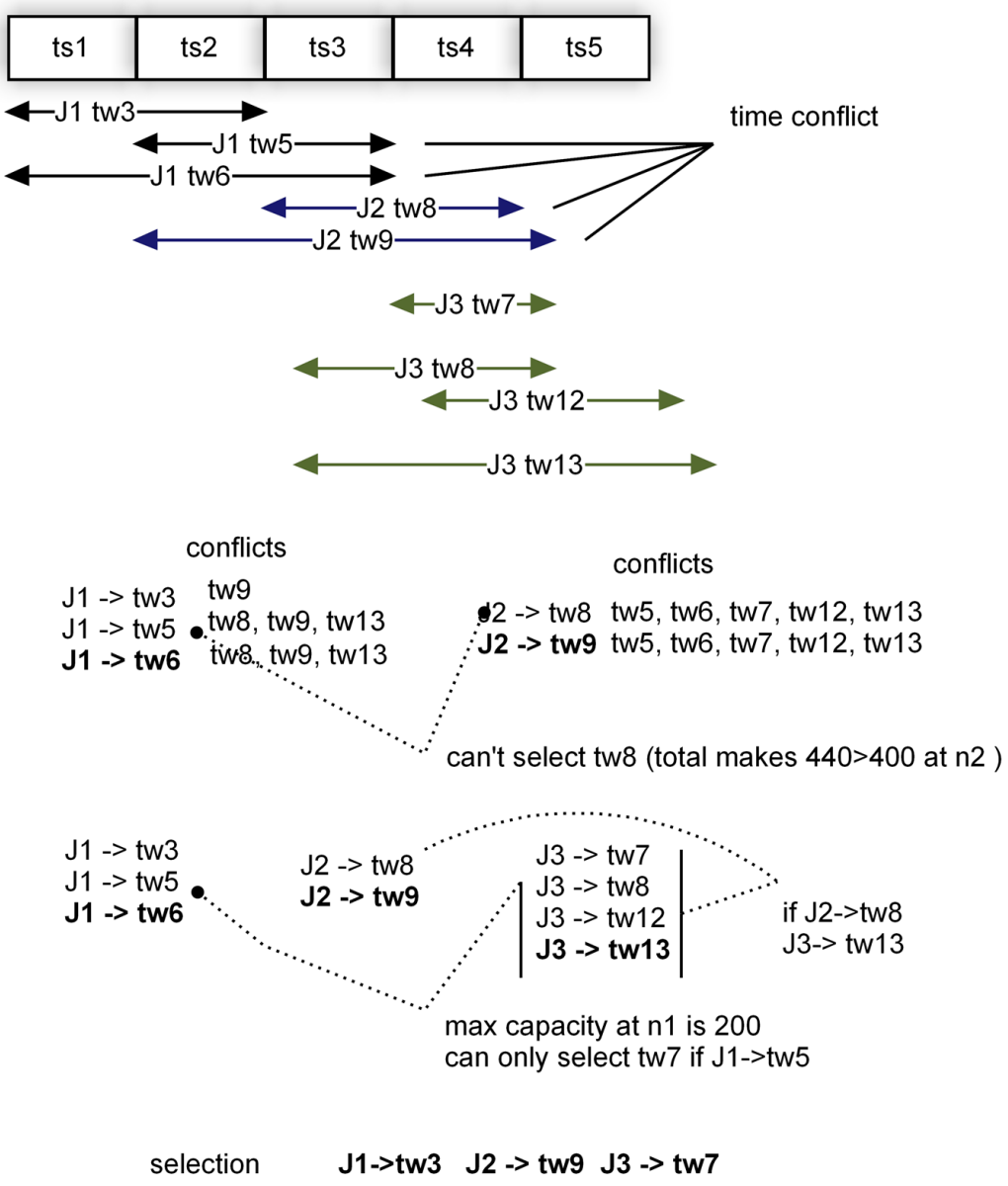



Figure 5.10: Assigning Jobs to Time Windows

# Chapter 6

# Scheduling with Advance Reservation

In this chapter, we propose an online scheduling algorithm such that the scheduler makes decision whenever a new data transfer job is submitted. We analyze data transfer scheduling between distributed resources with given time and resource constraints. Each job contains information about the total volume of data need to be transferred, source and destination end-points, and also the time period in which this data transfer operation need to be completed. Users submit data transfer jobs with a simple time constraint; an earliest time when this data will be ready to initiate the transfer, and a deadline when the user wants data transfer to be completed.

We first give a formal definition of the problem and we underline several technical characteristics of the target environment. We give a priority based greedy heuristic to manage the described scheduling problem with advance reservation and provisioning. We present the crucial decision points and give details about the proposed algorithm. We evaluate the given approach and show its implications in real life. We give design and implementation details of a practical data transfer scheduler with advance reservation.

## 6.1 A Scheduling Model with Reservation

The data transfer scheduler checks other jobs in the system and considers both time and resource conflicts. In order to admit a submitted job, it has to confirm the availability of resources to complete the transfer of the data before the given deadline. If a job has been admitted, a period of time is reserved in advance with required capacity in resources along the route between these two end-points. We consider that the scheduler has knowledge about the current and future capacity of resources that affect the end-to-end transfer performance. The users have the opportunity to give a desired period of time in which they want the transfer to be accomplished. If the scheduler cannot find a suitable time

slot, it tries to shift other jobs that had already have a reservation in the given time period without breaking any deadline requirements of previously admitted jobs.

One important constraint is to reserve contiguous time slots for each job such that a data transfer operation starts with fixed transfer rate and maintains this until the data volume has completely transferred. If the scheduler cannot reserve a proper time period for a submitted job, it either rejects the jobs or offers user another time frame which is found by extending the deadline. In such a case, we do not consider moving previously made reservations. This second phase in which the scheduler makes an offer by extending the deadline, encourages users to submit reasonable and appropriate deadlines when they are initially submitting the job.

The scheduler has two main objectives. First, it ensures that no other admitted job will be postponed due to making a new reservation. In addition to the fairness objective, it also tries to maximize the number of admitted jobs by moving reserved slots. The scheduler tries to be open a suitable period of time to admit a recently submitted job by resolving time and resource conflicts. On the other hand, it also selects time slots which gives earliest completion time and with minimum interference with other admitted jobs in the system.

## 6.2 Problem Definition

We define the topology as a time-dependent directed graph $G(V, E, T)$, with a node set $V$ of $n$ data transfer nodes, and an edge set $E \subseteq V$ with $m$ edges, where $e_k : (v_i, v_j)$ represents a connection from $v_i$ to $v_j$. For every connection between two nodes, there is a function of link capacity $u^{e_{ij}}(t)$ and $u^{e_{ji}}(t)$ where $t$ is a variable in time domain $T$. In addition to that, every node has separate upload and download capacities, $u_{out}^{v_i}$ and $u_{in}^{v_i}$ respectively.

We have a dynamic network environment in which edge capacities may vary over time. On the other hand, we know the maximum upload and download capacities in each data transfer node. We consider data transfer nodes as specialized machine(s), with back-end storage servers and data transfer

protocols, connected to the outside network with high-speed high-bandwidth interconnects.

A data transfer job is defined as $J_i = (v_i^s, v_i^d, M_i, t_i^E, t_i^L)$, where $M_i$ is the amount of data to be transferred from source $v_i$ to destination node $v_j$ within the time period of $(t_i^E, t_i^L)$. $t_i^E$ represents the earliest possible time when this data will be ready to start the transfer operation. $t_i^L$ represents the latest completion time the transfer operation needs to be finished.

$t_i^L$ defines a soft-deadline for the transfer operation such that the transfer operation is not interrupted if it cannot finish within the given deadline. However, the scheduler makes decisions according to the time constraints. It does not admit a job if it foresees that it is not possible to finish the transfer of the requested data before the given deadline.

## 6.3 Methodology

The scheduler tries to accomplish the task within the given time period but cannot promise a strict time guarantee due to the dynamic nature of the distributed environment. The transfer performance can be interrupted by external factors and also the operation can fail due to user and system errors. More information on failure awareness and managing errors will be explained further in the following sections.

If a submitted job is admitted, we set up a reservation for this job and allocate resources for a specific time period. A reservation is defined as $R_i = (v_i^s, v_i^d, \mu_i, t_i^s, t_i^e)$, where $\mu_i$ amount of bandwidth is reserved from source $v_i^s$ to $v_i^d$ between start time $t_i^s$ and end time $t_i^e$. A reservation request is only confirmed if there exits enough capacity satisfying the allocation of $\mu_i$ bandwidth in the given time period between $t_i^s$ and $t_i^e$. The total allocated bandwidth over the link $e_{ij}$ should be less than the capacity $u^{e_{ij}}(t)$ of the link for every instance of $t$ in $[t_i^s, t_i^e]$ . Similarly, the total in-coming bandwidth allocation should be less than $u_{in}^{v_i}$, and total out-going bandwidth allocation should be less than $u_{out}^{v_i}$, in the time period of $(t_i^s, t_i^e)$.

We consider non-preemptive operations where data transfer start at $t_i^s$ and continues till $t_i^e$ using

$\mu_i$ bandwidth from resources along the end-to-end route; consuming upload capacity of data transfer node $v_i$, link $e_{ij}$, and download capacity of $v_j$ . The duration of the time period, and the reserved bandwidth should be enough to satisfy transferring the requested amount of data. We simply say $M_i = \mu_i \times d_i$, where $d_i$ is the total time between $t_i^s$ and $t_i^e$. Therefore, it should satisfy the requested bandwidth during the entire period of time from start to end of this transfer. The problem is to find a contiguous set of time slots such that a fixed amount of bandwidth can be allocated to satisfy the data transfer job request.

We also would like to note that problem complexity is not bounded by given topology definition. Instead of defining the network as a directed graph, we can also consider an undirected network in which in-coming and on-going traffic share the total link capacity. Furthermore, we might have combined capacity in data transfer nodes where upload and download operations consume from same resource capacity in the nodes. Those specifications will affect the resource constraints. The given topology definition to describe environment has practical and implementation benefits. Further details will be explained in the following sections. On the other hand, our main concern is to solve time conflicts while arranging reservation for each admitted job.

## 6.4 Online Scheduling

We propose an online scheduler, so the first objective is to make a quick decision when a job is submitted. Therefore, we focus on polynomial scheduling algorithms which can easily be applied to instruct the underlying mechanism. One main objective of the scheduler is to maximize the number of admitted jobs. When a new job request arrives, the scheduler tries to open a reservation slot by moving previously made reservations in order to confirm an allocation and admit the new request. With such an objective in hand, one would expect the scheduler to accept jobs with small data volume and reject or delay jobs which have large data volume. Moreover, a job interfering with many other jobs and creating time conflicts will not be preferred. Those criteria will help maximize number of admitted jobs but will result in unfairness in practice. Therefore, a crucial objective is of our scheduler is ensure

that no other admitted job will be postponed due to making a new reservation for a new request.

The scheduler checks available time slots and considers resource constraints to find a proper allocation for the new request. If there is no availability, it tries to open a suitable period of time to admit a recently submitted job by moving previously made allocations to resolve time and resource conflicts.

If the scheduler cannot accept a job request within the given time period, it makes an offer to the user with a new latest completion time, $t_i^L$. In this case, when it fails to find a time slot at first, the scheduler does not try to move previous reservations. It searches for an available time slot without making an effort to optimize the placement of allocations. The scheduler notifies the user when this job can be completed earliest. If the user accepts, the job is admitted and the related time slots are reserved with required resource constraints. This policy encourages users to submit reasonable and appropriate deadlines with the initial submission. The scheduler will make an effort to find a time slot by trying to re-arrange the allocations. However, it fails to find a place, it will just offer the first available time period which might be sooner than the optimum.

The scheduler's other objectives are to complete the given request as early as possible, and to make a reservation which has minimum interference with other tasks. It selects time slots which gives earliest completion time and with minimum interference with other admitted jobs in the system. Even though there is available resource capacity both in nodes and the link, it is always beneficial not to have many concurrent transfers running at the same time. Furthermore, we would prefer to complete a job as soon as possible. We prefer allocating higher bandwidth for a shorter duration instead of allocating lower bandwidth for a longer duration. Data transfers which takes longer and which run on resources shared concurrently with other jobs, have higher failure probability. With multiple concurrent jobs sharing common resources, it is also becomes harder to estimate the transfer capacity. Therefore, we try to minimize the concurrency and we have preference to have minimum resource sharing. Additionally, we prefer to minimize the completion time of each job.

We would like emphasize the importance of giving a desired period of time in which we set the

earliest start and latest completion times to complete the transfer of a particular job request. The problem we define is quite different than other scheduling approaches with fixed start and end times given in the literature. Considering the time period between earliest start and latest completion times enables the scheduler to take into account several possible reservation options. This information given by the user allows us to make decisions considering other jobs, and find an allocation according to previously made reservations in the system. Overall, users have the flexibility to give a desired period of time instead of setting fixed start/end times for each job. It might seem that we are extending the problem and making the solution harder by setting a broader time space for each job, but this strategy has many benefits. By earliest start and latest completion times, we are not only providing a proper and practical approach in terms of real life requirements, we are also enabling the scheduler to move some previous allocations and open space for new requests, resulting in possible optimum reservations placement in an online scheduler.

## 6.5 Selection Criteria

We propose a new algorithm for online scheduling of data transfer jobs with advance reservation. Our approach, inspired from Gale-Shapley [105] and N-queen [96] algorithms, is to design an effective methodology which can easily be implemented and deployed in practice. One of the important criteria is to make quick and near optimum decisions despite the fact that the problem space is quite large. We present a novel methodology in which, for each job request, we assign a value of preference to the time reservations they could allocate. When a new job request cannot find a suitable time slot to make a reservation, it competes with previously admitted jobs to move their reservations and open a proper reservation time for itself. The outline of our scheduling methodology is as follows.

When a new request arrives, we first evaluate its time and resource constraints, and we try to find a reservation satisfying given criteria. We search through possible time allocations and make a reservation with the preference of selecting the one which gives earliest completion and shortest transfer duration. If there is no contiguous period of time with enough resource capacity in the given

search interval between $t^E$ and $t^L$, we start exploring possible options to move previously made time reservations to open a contiguous time slot that could satisfy the resource requirements of the new job request.

In this phase, we traverse each possible time allocations for the new request and look for jobs with less preference value for this time allocation. If there are jobs which have less preference for this time period, we select the job with minimum preference. We move out this job and take over its time allocation to make a temporary reservation for the new request. The job which recently moved out from its allocated time space starts competing with other jobs to find a new slot. This recursive operation continues until no reservation left to shift out.

**Preference:** Assigning a preference value is an important part in the design of the algorithm. Even though we assign random ranks to each transfer request, the algorithm in general will conclude with a scheduling decision. Since no previously admitted job will be displaced in order to allocate resources for a new request, we guarantee that scheduler will eventually satisfy users by making reservations based on their criteria. However, we have stated other objectives like minimizing the concurrency count and decreasing time to completion for a request. Therefore, we set the preference $p_{J_i}$ as $tw^{id}_{J_i}/tw^{num}_{J_i}$, where $tw^{id}$ is the current assigned time window, and $tw^{num}$ is the total number of time windows associated for this job. For a recently arrived job $tw^{id}$ represents the current time window we are evaluating to allocate. We compare its preference with other jobs already using this time window. A job with higher ratio is more close to its deadline; so, it has higher preference.

Alternatively, we can use $ts^{num}_{J_i}/ts^{total}_{J_i}$, where $ts^{num}$ is the total number of time steps in the current time window we are examining, and $ts^{total}$ is the total number of time steps this job can span over to make its reservation. This parameter encapsulates the concurrency count. A job assigned to a time window with higher preference has better chance to have its transfer overlapping with other transfer operations. Recall that time steps represent a static network structure. Therefore, we prefer to assign a job to a time window which includes less time steps. Therefore, we favor a job which has already been using more time steps compared to the total number of time steps it can cover.

On the other hand, the first preference metric also considers the concurrency count. Since we search time windows in a particular order, this preference metric of $tw^{id}_{J_i}/tw^{num}_{J_i}$ convince our objectives. While searching time windows, we consider to traverse earliest one, but we also look at shorter ones first (which are including less time steps). Therefore, the preference metric better fits by both minimizing concurrency count and time to completion.

## 6.6 Evaluation

A new data transfer request is only admitted only if we could allocate time and resource capacity in advance without breaking the constraints of previously admitted jobs. At the end, if we can still find a space for all previously admitted jobs and the new request, we admit the new request and make the temporarily made reservation permanent. Otherwise, we roll back all temporary reservations and return back to the previous state. We try and execute the same search procedure for other possible time allocations that this new request can reserve.

If we succeed in none of them, we could not end up finding a schedule satisfying all admitted job and this new request, we either reject the new request or suggest a new latest completion time. We simply inform the user that we could not find a time period in the given search interval to accomplish the data transfer operation. In this case, the scheduler looks forward and selects a time period which gives earliest completion for the request without trying to shift previously made reservations.

The following Algorithm gives a glimpse of the algorithm used to search and find a new resource to job mapping, in order to admit a new job, and displace a previously admitted jobs if necessary to open space for the new request.

For each new job, we divide the search interval into time steps. The search interval $[t^E, t^L]$ of a job is the time period between earliest start time $t^E$ and latest completion time $t^L$ in which the data need to be transmitted. A time step represents the longest duration of time in which we have a stable discrete status in terms of available bandwidth over the link and data transfer nodes. The

**Input**: A set of admitted jobs(already in the system) and their active advance Allocations
**Input**: A new a job request $J_i$ with earliest start $t^E$, latest completion $t^L$, volume $M$, source $v^s$ and destination $v^d$
**Output**: A new set of allocation If this job is admitted
Get all time windows which contains all time steps between $t^E$ and $t^L$ ;
Obtain static bandwidth availability snapshot for $v^s$ , $v^d$ and the link $v^s$ $v^d$ ;
Search time windows to find a reservation satisfying given criteria ;
*Time window search sequence :* ;
*(According to earliest completion time but time window with shortest duration preferred first)*;
**if** *A suitable time window satisfying resource constraints found* **then**
    Make allocation and admit the job;
**else**
    **for** *all time windows that could satisfy new job request* **do**
        For time window *tw*, search if there is a job with less preference;
        *Omit jobs which are flagged not to be displaced* ;
        *Request already started or already displaced in the current search sequence are omitted*;
        **if** *such a job $J_k$ is found* **then**
            Displace $J_k$ and make a reservation for $J_i$;
            $J_k$ and $J_i$ are flagged ;
            Run **Scheduling Algorithm** for $J_k$ to find a new reservation for $J_k$ ;
**if** *all jobs in the system (including $J_i$) ) are scheduled* **then**
    Admit the new job $J_i$;
    Accept the new allocation (resource-to-job) mapping by committing the final Reservation Set;
**else**
    Rollover to the initial state (resource-to-job mapping);
**Scheduling Algorithm:** search pattern to find resource and schedule a new request

set of confirmed reservations in the system characterize time steps since they change the available bandwidth values in the topology. If there are $r$ committed reservations falling into the period, there can be maximum $2r + 1$ different time steps in the worst-case. If $s$ is the total number of time steps, there are $(s \times (s + 1))/2$ time windows since time windows are subsequent combinations of time steps. We search through these time windows in a sequential order to check whether we can satisfy the requested allocation in that time window. Assume that there are already $n$ jobs in the system which have already been admitted. When we receive the $(n + 1)^{th}$ job, and we could not confirm a reservation just by looking time windows it can span over, we try to displace other jobs to open space for this operation. We sequentially traverse time windows that can satisfy given criteria, and try to find a job with less preference that already has allocation in the time window we are considering.

As it has been described above, this recursive process will end when we cannot place a previously admitted job. Therefore, there can be maximum $n$ tries. Thus, total complexity is bounded by number of jobs and number of time steps, $O(n \times s^2)$. Time steps are associated with reservations and the total number linearly scales with the number of reservations in system. In a very extreme case where all jobs fall into same search interval, complexity is $O(n^3)$ such that every admitted job has a reservation committed.

## 6.7 Implementation Details

Another challenge in designing an advance reservation system is to find an appropriate data structure to keep bandwidth availability in a time-dependent network. The common approach presented in the literature is to divide the entire time period into time slots and store available bandwidth over a link for each time slot. When a new reservation is committed and added to the system, we proceed and update bandwidth availability and time slot information for every link on the allocated path. Using such a technique, in which we accumulate resource availability for time slots for every link in a network, enables straightforward evaluation since all resource availability has been pre-computed. On the other hand, the total data size increases dramatically especially for large network with many reservations committed. There have been several studies analyzing data structures for network routing with advance reservation [28, 112, 110]. Further, we need an effective methodology which we can also benefit in calculating static snapshot graphs $G(t_w)$, during maximum bandwidth calculation and time window evaluation. Since we already have reservation information, which includes path information and allocated bandwidth value, we can automatically generate the bandwidth availability for all links for a specific time period if we know reservations that are active in the time window we are evaluating. Simply, we deduct allocated bandwidth of a reservation from the available bandwidth of a link if the path for this reservation uses the link we consider.

We present a specialized linked-list data structure that holds time steps and a set of active reservation identifiers associated for each time step. This information is updated on the fly. When a new

reservation is committed, the data structure is updated and new time steps are added automatically if necessary. If a reservation is canceled, its identifier is removed from the set of reservations that belongs to time steps the canceled reservation spans over. The main purpose of this data structure is to query time windows quickly and retrieve list of active reservations in time windows. We only need time steps that fall into the given search interval. Time steps are indexed for faster operations and a set of reservations is returned for each time window.

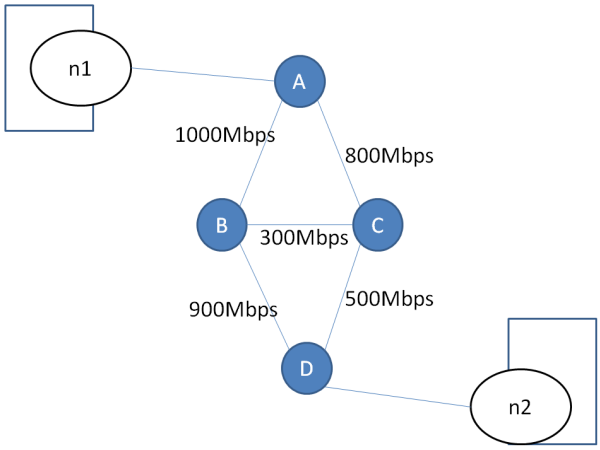



Figure 6.1: Connection between two data nodes

The connection between two end-point may span over multiple routers. As can be seen in Figure 6.1, data nodes are connected to the edge-routers in a network. Performance of searching bandwidth allocation between two edge-routers and finding possible network reservation options are crucial in designing a scheduler with advance reservation and provisioning. We have tested the network reservation algorithm presented in previous chapters by generating large random graphs. Random requests with user parameters such as data volume, earliest start and end times are generated for reservations within a 200hrs time interval. Figure 6.2 and 6.3 show overall performance with various number of reservations applied in each case. Each point in the graph is average of 100 measurements. Those tests are performed in mid-range workstation, and we were able to process and search all related time windows to find a reservation in a timely manner. Note that it is very unlikely in real life to have thousands of committed reservations in a short period. Furthermore, we would have a maximum hop count parameter in real-life. We ignored the maximum hop count parameter and set it to infinite in order to evaluate performance in large and complex system.

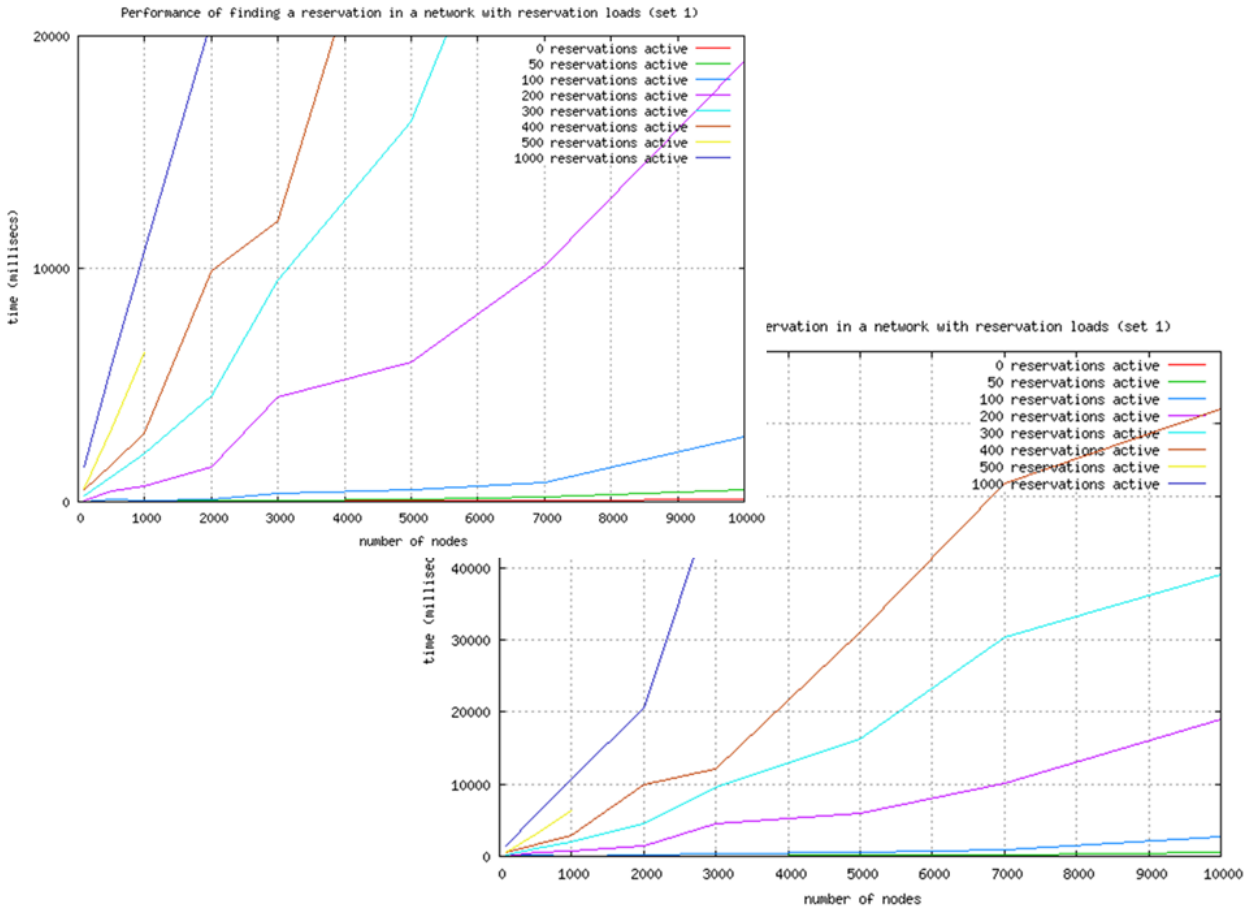



Figure 6.2: Search performance to find a reservation in a network (Set 1: sparse graph with less cycles)

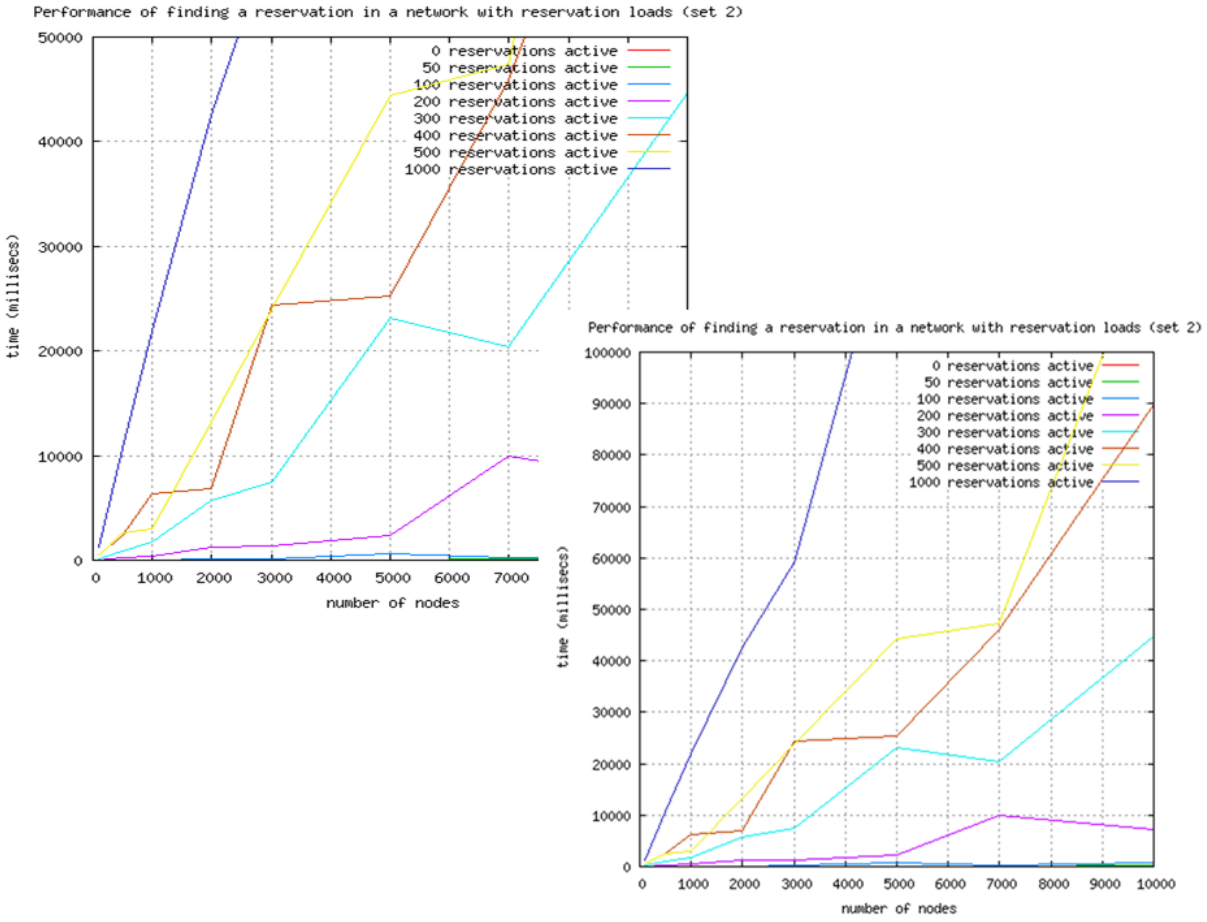



Figure 6.3: Search performance to find a reservation in a network (Set 2: dense graph with more cycles)

# Chapter 7

# Conclusion

In this dissertation, we developed a new data transfer scheduling paradigm in which data movement operations are scheduled in advance. This improves current systems by allowing researchers/users and higher level meta-schedulers to use data placement as a service where they can plan ahead and reserve the scheduler time in advance for their data movement operations. In general, the number of reservation options is exponential with the number of nodes $n$, and current reservation commitments. We present a novel approach for path finding in time-dependent networks taking advantage of user-provided parameters of total volume and time constraints. The network reservation algorithm has already been implemented for integration into future versions of OSCARS. On the other hand, network provisioning is not sufficient by itself for end-to-end high performance data transfer. In order to take advantage of the available network bandwidth, client sites should provision other resources such as storage capacity and bandwidth. Therefore, we have studied scheduling data transfer operation with resource and time conflicts. We have developed a new scheduling methodology considering resource allocation in client sites and bandwidth allocation on network link connecting resources. Our methodology provides a basis for provisioning end-to-end high performance data transfers in which users submit their jobs with time and resource constraints to make an advance schedule. Our future work includes implementation and integration of the online scheduling algorithm with advance reservation. We also benefit from some important features such as early error detection, job aggregation, failure awareness, network bandwidth allocation and dynamic adaptation. Some other contributions of those techniques include enhanced reliability, adaptability, and performance optimization of scheduling distributed data placement tasks. Results presented in this dissertations will also contribute to the research community by analyzing resource allocation and management, fault tolerance in data movement, and coordination of bulk data transfers in data intensive computing.

# Bibliography


[1] Advance Network Testbed for Research and Development. http://www.jgn.nict.go.jp/english/.

[2] Dynamic Resource Allocation via GMPLS Optical Networks. http://dragon.maxgigapop.net/.

[3] Energy Sciences Network. http://www.es.net.

[4] High speed TransAtlantic network for the LHC Community. http://lhcnet.caltech.edu/.

[5] Internet2. www.internet2.edu.

[6] Louisiana Optical Network Initiative. http://www.loni.org/.

[7] National LamdaRail. http://www.nlr.net/.

[8] OSCARS: On-demand secure circuits and advance reservation system. www.es.net/oscars.

[9] Stork 1.0. http://www.cct.lsu.edu/~balman/stork/downloads.

[10] GridFTP : Developer's Guide. http://www.globus.org, 2007.

[11] Ismail Akturk and Mehmet Balman et al. Distributed Data Sharing with PetaShare for Collaborative Research in CyberTools. In Proceedings of NSF EPSCoR RII Symposium, 2009.

[12] G. Aldering and SNAP Collaboration. Overview of the supernova/acceleration probe (snap). http://www.citebase.org/abstract?id=oai:arXiv.org:astro-ph/0209550, 2002.

[13] Bill Allcock, Ian Foster, Veronika Nefedova, Ann Chervenak, Ewa Deelman, Carl Kesselman, Jason Lee, Alex Sim, Arie Shoshani, Bob Drach, and Dean Williams. High-performance remote access to climate simulation data: A challenge problem for data grid technologies. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 46–46, New York, NY, USA, 2001. ACM Press.

[14] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in highperformance computational grid environments. *Parallel Computing. 2001.*, 2001.

[15] M. Allen and R. Wolski. The livny and plank-beck problems: Studies in data movement on the computational grid. In *Supercomputing 2003*, November 2003.

[16] Dragos Andrei, Massimo Tornatore, Dipak Ghosal, Charles Martel, and Biswanath Mukherjee. Provisioning data-aggregation sessions in lambda grids. Technical report, US Davis, CSE, 2008.

[17] Yossi Azar and Oded Regev. Strongly Polynomial Algorithms for the Unsplittable Flow Problem. In *Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization (IPCO*, pages 15–29, 2001.

[18] Mehmet Balman. Failure-Awareness and Dynamic Adaptation in Data Scheduling. *M.S. Thesis, Louisiana State University*, 2008.

[19] Mehmet Balman, Ismail Akturk, and Tevfik Kosar. Intermediate Gateway Service to Aggregate and Cache I/O operations into Data Repositories", howpublished=.

[20] Mehmet Balman, Ismail Akturk, S. Roy, T. Kosar, G. Allen, and S. Acharya. Data Migration in Distributed Repositories for Collaborative Research. In Proceedings of NSF EPSCoR RII Symposium, 2009.

[21] Mehmet Balman, Evangelos Chaniotakis, Arie Shoshani, and Alex Sim. Advance Network Reservation and Provisioning for Science. (Abstract), UK e-science All-hands Meeting, 2009.

[22] Mehmet Balman and Tevfik Kosar. Dynamic Adaptation of Parallelism Level in Data Transfer Scheduling. *International Workshop on Adaptive Systems in Heterogeneous Environments (ASHEs 2009), Fukuoka, Japan*, 2009.

[23] Mehmet Balman and Tevfik Kosar. Early Error Detection and Classification in Data Transfer Scheduling. *Proceedings of International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC-2009), Fukuoka, Japan*, 2009.

[24] Mehmet Balman and Tevfik Kosar. Data Scheduling for Large Scale Distributed Applications. In *the 5th ICEIS Doctoral Consortium, In conjunction with the International Conference on Enterprise Information Systems (ICEIS'07)*. Funchal, Madeira-Portugal, June, 2007.

[25] Mehmet Balman, Ibrahim Suslu, and Tevfik Kosar. Distributed Data Management with PetaShare. Poster presentation, The 15th ACM SIGAPP Mardi Gras Conference, 2008.

[26] Nikhil Bansal, Zachary Friggstad, Rohit Khandekar, and Mohammad R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 702–709, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[27] Uyless N. Black. *MPLS and Label Switching Networks (2nd Edition)*. Printece-Hall series on advance communicaiton technologies.

[28] Lars-Olof Burchard. Analysis of data structures for admission control of advance reservation requests. *IEEE Trans. on Knowl. and Data Eng.*, 17(3):413–424, 2005.

[29] Lars-Olof Burchard. Networks with advance reservations: Applications, architecture, and performance. *J. Netw. Syst. Manage.*, 13(4):429–449, 2005.

[30] R. Buyya and S. Venugopal. The gridbus toolkit for service oriented grid and utility computing: An overview and status report. In *1st IEEE Int. Workshop Grid Economics and Business Models (GECON 2004.*, 2004.

[31] Alexandra Carpen-Amarie, Mugurel Andreica, and Valentin Cristea. An algorithm for file transfer scheduling in grid environments. http://arxiv.org/pdf/0901.0291, 2009.

[32] Cern. The world's largest particle physics laboratory. http://public.web.cern.ch, 2006.

[33] cFlowd. Traffic Flow Analysis Tool. http://www.caida.org/tools/measurement/cflowd/, 2006.

[34] Ismail Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Records*, 1645:170–175, 1998.

[35] Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation algorithms for the unsplittable flow problem. In *APPROX '02: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 51–66, London, UK, 2002. Springer-Verlag.

[36] William C. Cheng, Cheng fu Chou, Leana Golubchik, Samir Khuller, and Yung-Chun (Justin) Wan. Large-scale data collection: a coordinated approach. In *in Proceedings of IEEE INFOCOM*, pages 218–228, 2003.

[37] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers in a distributed network. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 254–266, New York, NY, USA, 1983. ACM.

[38] R. Cohen, N. Fazlollahi, and D. Starobinski. Graded channel reservation with path switching in ultra high capacity networks. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pages 1–10, Oct. 2006.

[39] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Infromation Services for Distributed Resource Sharing. In *Proc. of the 10th IEEE High Performance Distributed Computing*, pages 181–184, 2001.

[40] Sang Son Department and Sang H. Son. A priority-based scheduling algorithm for real-time databases.

[41] Bolin Ding, Jeffrey Xu Yu, and Lu Qin. Finding time-dependent shortest paths over large graphs. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 205–216, New York, NY, USA, 2008. ACM.

[42] Eduard Escalona, Salvatore Spadaro, Jaume Comellas, and Gabriel Junyent. Advance reservations for service-aware gmpls-based optical networks. *Comput. Netw.*, 52(10):1938–1950, 2008.

[43] Guy Even, Magnús M. Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *J. of Scheduling*, 12(2):199–224, 2009.

[44] Barak Farzad, Neil Olver, and Adrian Vetta. A priority-based model of routing, 2008.

[45] FastTCP. An alternative congestion control algorithm in tcp. http://netlab.caltech.edu/FAST/, 2006.

[46] FDT. Fast Data Transfer. http://monalisa.cern.ch/FDT.

[47] I. Foster and C. Kesselman. The Globus Project: A Status Report. In *7th IEEE Heterogeneous Computing Workshop (HCW 98)*, pages 4–18, March 1998.

[48] Ian Foster, Carl Kesselman, Craig Lee, Bob Lindell, Klara Nahrstedt, and Alain Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *In Proceedings of the International Workshop on Quality of Service*, pages 27–36, 1999.

[49] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource allocation in the grid using reinforcement learning. *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 3:1314–1315, 2004.

[50] S. Ganguly, A. Sen, G. Xue, B. Hao, and B. Shen. Optimal routing for fast transfer of bulk data files in time-varying networks. IEEE Int. Conf. on Communications, 2008.

[51] Mordecai J. Golin. Bipartite matching and the hungarian method. Course Notes, Hong Kong University of Science and Technology.

[52] GridFTP. Protocol extensions to ftp for the grid. http://www.globus.org, 2006.

[53] R. Guerin and A. Orda. Networks with advance reservations: the routing perspective. INFOCOMM 2000, 2000.

[54] Chin Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston. Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pages 1–8, Oct. 2006.

[55] Tony Hey and Anne Trefethen. The Data Deluge: An e-science perspective. *Grid Computing: Making the Global Infrastructure a Reality. Chichester, UK: John Wiley & Sons, Ltd.*, pages 809–824, 2003.

[56] Sandy Irani and Vitus Leung. Scheduling with conflicts on bipartite and interval graphs. *J. of Scheduling*, 6(3):287–307, 2003.

[57] Edward G. Coffman Jr., M. R. Garey, David S. Johnson, and Andrea S. LaPaugh. Scheduling file transfers. *SIAM J. Comput.*, 14(4):743–780, 1985.

[58] Eun-Sung Jung, Yan Li, Sanjay Ranka, and Sartaj Sahni. An evaluation of in-advance bandwidth scheduling algorithms for connection-oriented networks. In *ISPAN '08: Proceedings of the The International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 133–138, Washington, DC, USA, 2008. IEEE Computer Society.

[59] Jon M. Kleinberg. Single-source unsplittable flow. In *In Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 68–77, 1996.

[60] G. Kola, T. Kosar, J. Frey, M. Livny, R. Brunner, and M. Remijan. Disc: A system for distributed data intensive scientific computing. In *Proc. of First Workshop on Real, Large Distributed Systems. San Francisco, CA, December 2004.*, 2004.

[61] Stavros G. Kolliopoulos and Clifford Stein. Improved approximation algorithms for unsplittable flow problems (extended abstract). In *In Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 426–435, 1997.

[62] Petr Kolman. A note on the greedy algorithm for the unsplittable flow problem. *Information Processing Letters*, 88(3):101 – 105, 2003.

[63] T. Kosar, Mehmet Balman, I. Suslu, E. Yildirim, and D. Yin. Data-Aware Distributed Computing with Stork Data Scheduler. In Proceedings of SEE-GRID-SCI'09, 2009.

[64] T. Kosar and M. Livny. Stork: Making Data Placement a first class citizen in the grid. In *In Proceedings of the 24th Int. Conference on Distributed Computing Systems, Tokyo, Japan, March 2004.*, 2004.

[65] Tevfik Kosar. Data placement in widely distributed systems. *Ph.D. Thesis, University of Wisconsin-Madison*, 2005.

[66] Tevfik Kosar and Mehmet Balman. A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems*, Vol.25 No.4, pp.406-413, 2009.

[67] Tevfik Kosar, George Kola, and Miron Livny. Data pipelines: enabling large scale multi-protocol data transfers. In *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, pages 63–68, 2004.

[68] E. Laure. The EU datagrid setting the basis for production grids. *Journal of Grid Computing. Springer*, 2(4), December 2004.

[69] Zhaoming Li, Qiang Song, and Ibrahim Habib. Cheetah virtual label switching router for dynamic provisioning in ip optical networks. *Optical Switching and Networking*, 5(2-3):139–149, 2008. Advances in IP-Optical Networking for IP Quad-play Traffic and Services.

[70] Yunyue Lin and Qishi Wu. On design of bandwidth scheduling algorithms for multiple data transfers in dedicated networks. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 151–160, New York, NY, USA, 2008. ACM.

[71] Yunyue Lin and Qishi Wu. On design of bandwidth scheduling algorithms for multiple data transfers in dedicated networks. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 151–160, New York, NY, USA, 2008. ACM.

[72] Bertram Ludscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

[73] Ravi K. Madduri, Cynthia S. Hood, and William E. Allcock. Reliable file transfer in grid environments. In *LCN '02: Proceedings of the 27th Annual IEEE Conference on Local Computer Networks*, pages 737–738, Washington, DC, USA, 2002. IEEE Computer Society.

[74] Florin Manea and Calina Ploscaru. Solving a combinatorial problem with network flows, 2004.

[75] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[76] Mehmet Balman and Tevfik Kosar. From Micro- to Macro-processing: A Generic Data Management Model. In *Poster presentation, The 8th IEEE/ACM International Conference on Grid Computing*. (Grid2007)Austin, Sept 2007.

[77] NetFlow. Cisco ios netflow. http://www.cisco.com, 2006.

[78] NWS. NWS: Network Weather Service. http://nws.cs.ucsb.edu/ewiki/, 2006.

[79] The Office of Science Data-Management Challenge. Report from the doe office of science data-management workshops, March-May 2004.

[80] OptorSIM. A grid simulator. http://www.gridpp.ac.uk/demos/optorsimapplet/, 2006.

[81] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3):607–625, 1990.

[82] Marek Piotrów. A note on constructing binary heaps with periodic networks. *Inf. Process. Lett.*, 83(3):129–134, 2002.

[83] J. S. Plank, S. Atchley, Y. Ding, and M. Beck. Algorithms for high performance, wide-area distributed file downloads. *Parallel Processing Letters*, 13(2):207–224, June 2003.

[84] Kannan Rajah, Sanjay Ranka, and Ye Xia. Scheduling bulk file transfers with start and end times. *Comput. Netw.*, 52(5):1105–1122, 2008.

[85] Kavitha Ranganathan and Ian Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*, page 352, Washington, DC, USA, 2002. IEEE Computer Society.

[86] Kavitha Ranganathan and Ian Foster. Computation scheduling and data replication algorithms for data grids. *Grid resource management: state of the art and future trends*, pages 359–373, 2004.

[87] Kavitha Ranganathan and Ian T. Foster. Simulation studies of computation and data scheduling algorithms for data grids. *J. Grid Comput.*, 1(1):53–62, 2003.

[88] N. S. V. Rao, W. R. Wing, S. M. Carter, and Q. Wu. Ultrascience net: network testbed for large-scale science applications. *Communications Magazine, IEEE*, 43(11):S12–S17, 2005.

[89] N.S.V. Rao, Qishi Wu, Song Ding, S.M. Carter, W.R. Wing, A. Banerjee, D. Ghosal, and B. Mukherjee. Control plane for advance bandwidth scheduling in ultra high-speed networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–5, April 2006.

[90] Biju K Raveendran, Sundar Balasubramaniam, and S Gurunarayanan. Evaluation of priority based real time scheduling algorithms: choices and tradeoffs. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 302–307, New York, NY, USA, 2008. ACM.

[91] Sartaj Sahni, Nageshwara Rao, Sanjay Ranka, Yan Li, Eun-Sung Jung, and Nara Kamath. Bandwidth scheduling and path computation algorithms for connection-oriented networks. In *ICN '07: Proceedings of the Sixth International Conference on Networking*, page 47, Washington, DC, USA, 2007. IEEE Computer Society.

[92] Olov Schelen and Stephen Pink. An agent-based architecture for advance reservations. *Local Computer Networks, Annual IEEE Conference on*, 0:451, 1997.

[93] SDSC. San Diego Supercomputer Center iRODS project. https://www.irods.org/, 2008.

[94] Farhad Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, 1990.

[95] Arie Shoshani, Alexander Sim, and Junmin Gu. *Storage Resource Managers: Essential Components for the Grid*. Kluwer Academic Publishers, 2003.

[96] Rok Sosic and Jun Gu. A polynomial time algorithm for the n-queens problem. *SIGART Bull.*, 1(3):7–11, 1990.

[97] Sebastien Soudan, Bin Bin Chen, and Pascale Vicat-Blanc Primet. Flow scheduling and endpoint rate control in gridnetworks. *Future Gener. Comput. Syst.*, 25(8):904–911, 2009.

[98] sTCP. Scalable TCP. http://www.deneholme.net/tom/scalable/, 2006.

[99] Jon M. Stokes. *Inside the Machine - An Illustrated Introduction to Microprocessors and Computer Architecture*. No Starch Press, December 2006, 320 pp.

[100] Stream. Stream Control Transmission Protocol. http://tools.ietf.org/html/rfc2960, 2006.

[101] Ibrahim Suslu, Fatih Turkmen, Mehmet Balman, and Tevfik Kosar. Choosing Between Remote I/O versus Staging in Large Scale Distributed Applications. in Proceedings of ISCA 21st Int. Conference on Parallel and Distributed Computing and Applications, 2008.

[102] Savera Tanwir, Lina Battestilli, Harry Perros, and Gigi Karmous-Edwards. Dynamic scheduling of network resources with advance reservations in optical grids. *Int. J. Netw. Manag.*, 18(2):79–105, 2008.

[103] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. Visual Grid Workflow in Triana. *Journal of Grid Computing*, 3(3-4):153–169, September 2005.

[104] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The Triana Workflow Environment: Architecture and Applications. In Ian Taylor, Ewa Deelman, Dennis Gannon, and Matthew Shields, editors, *Workflows for e-Science*, pages 320–339, Secaucus, NJ, USA, 2007. Springer, New York.

[105] Chung-Piaw Teo, Jay Sethuraman, and Wee-Peng Tan. Gale-shapley stable marriage problem revisited: Strategic issues and applications. *Manage. Sci.*, 47(9):1252–1267, 2001.

[106] Douglas Thain, Todd Tannenbaum, and Miron Linvy. Grid Computing: Making the Global Infrastructure a Reality. In *Condor and the Grid*, number ISBN:0-470-85319-0, pages 299–336. John Wiley, 2003.

[107] J. A. Tyson. Large Synoptic Survey Telescope: Overview. In J. A. Tyson and S. Wolff, editors, *Survey and Other Telescope Technologies and Discoveries. Edited by Tyson, J. Anthony; Wolff, Sidney. Proceedings of the SPIE, Volume 4836, pp. 10-20 (2002).*, pages 10–20, December 2002.

[108] M. Veeraraghavan, H. Lee, E.K.P. Chong, and H. Li. A varying-bandwidth list scheduling heuristic for file transfers. In *Communications, 2004 IEEE International Conference on*, volume 2, pages 1050–1054 Vol.2, June 2004.

[109] Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, pages 75–80, 2004.

[110] Tao Wang and Jianer Chen. Bandwidth tree - a data structure for routing in networks with advanced reservations. In *PCC '02: Proceedings of the Performance, Computing, and Communications Conference, 2002. on 21st IEEE International*, pages 37–44, Washington, DC, USA, 2002. IEEE Computer Society.

[111] X. Wang, D. Huang, I. Akturk, Mehmet Balman, G. Allen, and T. Kosar. Semantic Enabled Metadata Management in PetaShare. International Journal of Grid and Utility Computing (IJGUC), 2009.

[112] Qing Xiong, Chanle Wu, Jianbing Xing, :ibing Wu, and Huyin Zhang. A linked-list data structure for advance reservation admission control. *Networking and Mobile Computing*, 2005.

[113] Esma Yildirim, Mehmet Balman, and Tevfik Kosar. Dynamically tuning level of parallelism in wide area data transfers. In *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 39–48, New York, NY, USA, 2008. ACM.

[114] J. Zheng, B. Zhang, and H.T. Mouftah. Toward automated provisioning of advance reservation service in next-generation optical internet. pages 68–74. IEEE Communications Magazine, 2006.

# Vita

Mehmet Balman is currently a computer engineer in the High Performance Computing Research Department of the Computational Research Division at Lawrence Berkeley National Laboratory in Berkeley, California. He has been working in the Scientific Data Management Research and Development Group at LBNL since 2009. He has been pursuing his doctoral degree in computer science at Louisiana State University since 2005. He received his bachelor of science and master of science degrees in computer engineering from Bogazici University, Istanbul, Turkey. He also holds a master of science degree in system sciences from the Department of Computer Science at Louisiana State University. He got exposed to several years of industrial experience as system administrator and R&D specialist, at various software companies before joining LSU. He also worked as a summer intern in Los Alamos National Laboratory and Lawrence Berkeley National Laboratory. During his study at LSU, he worked as a Teaching Assistant in the Department of Computer Science, and as a Research Assistant in the Center for Computation & Technology. He was involved in the development of Stork data placement scheduler, and PetaShare project. His research interests include high performance scientific computing, distributed data management, network management, and job scheduling in widely distributed systems.