2008

Failure-awareness and dynamic adaptation in data scheduling

Mehmet Balman
Louisiana State University and Agricultural and Mechanical College

FAILURE-AWARENESS AND DYNAMIC ADAPTATION IN DATA SCHEDULING

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
College of Basic Sciences
in partial fulfillment of the
requirements for the degree of
Master of Science in Systems Science

in

The Department of Computer Science

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

Table of Contents

Abstract

Over the years, scientific applications have become more complex and more data intensive. Especially large scale simulations and scientific experiments in areas such as physics, biology, astronomy and earth sciences demand highly distributed resources to satisfy excessive computational requirements. Increasing data requirements and the distributed nature of the resources made I/O the major bottleneck for end-to-end application performance. Existing systems fail to address issues such as reliability, scalability, and efficiency in dealing with wide area data access, retrieval and processing. In this study, we explore data-intensive distributed computing and study challenges in data placement in distributed environments. After analyzing different application scenarios, we develop new data scheduling methodologies and the key attributes for reliability, adaptability and performance optimization of distributed data placement tasks. Inspired by techniques used in microprocessor and operating system architectures, we extend and adapt some of the known low-level data handling and optimization techniques to distributed computing. Two major contributions of this work include (i) a failure-aware data placement paradigm for increased fault-tolerance, and (ii) adaptive scheduling of data placement tasks for improved end-to-end performance. The failure-aware data placement includes early error detection, error classification, and use of this information in scheduling decisions for the prevention of and recovery from possible future errors. 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.

Chapter 1
Introduction

Over the years, scientific applications have become increasingly data intensive. Computationally intensive science (*e-Science*), which has wide application areas including particle physics, bio-informatics and social simulations, demands highly distributed networks and deals with immense data sets [64, 39]. Besides being excessively challenging computationally, these applications have become increasingly data intensive and gradually went from giga- to peta-scale [52]. Especially, scientific experiments using geographically separated and heterogeneous resources necessitated transparently accessing distributed data and analyzing huge collection of information [1, 38, 66]. Hence, many recent studies investigate new approaches for data management and data transfer in distributed systems [33, 107].

Data management has remained one of the crucial problems in every stage of computer engineering, 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, both in operating system design and in microprocessor architecture [86]. We study how similar challenges are handled in other layers of computing systems such as microprocessor and operating systems. 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 [1].

Even in the very recent multi-core era, importance of data access and data management cannot be overemphasized. Since small and large scale systems have similar problems, we can utilize approaches in microprocessor and operating system kernel architecture to come up with a broader perspective in which we can extend known methodologies to be used in distributed systems. Hence, the importance of data management research in computer science, especially on large-scale systems, is deeply felt.

On the other hand, data transfer in microprocessors through high speed bus is deterministic in most cases. The scenario in distributed systems is quite different. Data transfer in a distributed environment is prone to frequent failures resulting from back-end system level problems, like connectivity failure which

[1] A supercomputer is a device for turning compute-bound problems into I/O-bound problems - Seymour Cray

is technically untraceable by users. Error messages are not logged efficiently, and sometimes are not relevant/useful from user's point-of-view. Our study explores the possibility of an efficient error detection and reporting system for such environments. Besides, early error detection and error classification have great importance in organizing data placement jobs. It is necessary to have well-defined error detection and error reporting methods to increase the usability and serviceability of existing data transfer protocols and data management systems. 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.

The rest of this thesis is outlined as follows.

In Chapter 2, we define data-intensive distributed computing and investigate data scheduling methodologies. We analyze different scenarios for possible use cases of data placement tasks. And, we identify parameters affecting data transfer to discover key attributes for adaptability and optimization in terms of performance. We explain approaches to integrate data placement into job management systems and discuss methodologies to incorporate with other middleware tools.

In Chapter 3, we focus on problems on accessing and storing data that have been encountered in the early phases of processor design, and also analyze how those issues have been resolved to enable today's fast and efficient microprocessors. We present a data-aware distributed computing paradigm underlining the fact that different layers of computing systems have similarities in data and CPU management subsystems.

In Chapter 4, we present an adaptive approach for organizing data placement jobs. Inspired from prefetching and caching techniques to overcome latency bottleneck between interconnects in microprocessor design, we study several levels of aggregation for performance optimization. First, we explain dynamic tuning of number of parallel streams which is simply aggregation of TCP connections. Then, we describe effects of concurrent transfer jobs running simultaneously. We propose an adaptive methodology for setting parallelism level in data transfers such that data placement scheduler dynamically tunes parameters without depending on any external profilers. Later, we describe aggregation of multiple jobs in which data transfer jobs are combined and managed as a single job by the scheduler to eliminate the connection cost of underlying transfer protocol.

In Chapter 5, we study early error detection and error classification, and also failure and performance issues when transfer is in progress. We propose an error reporting framework and a failure-aware data

transfer life cycle. We investigate the applicability of proposed error detection and classification techniques to improve arrangement of data placement jobs and to enhance decision making process of data placement schedulers.

We explain implementation details and discuss evaluation results from our experiments. Finally, we conclude with future work and possible research problems in data placement scheduling in distributed computing area.

Chapter 2
Scheduling Data Placement Jobs

There are several studies concentrating on data management in scientific applications [31, 33, 107]; however, resource allocation and job scheduling considering the data requirements still remains as an open problem. Data placement is a part of the job scheduling dilemma and it should be considered as a crucial factor affecting resource selection and scheduling in distributed computing [77, 79, 19, 2].

In this chapter, we focus on data-intensive distributed computing and describe the data scheduling approach to manage large scale scientific and commercial applications. We identify structures in the overall model and also analyze different scenarios for possible use cases of data placement tasks to discover key attributes affecting performance.

We start with data placement operations inside a single host without network interference. We proceed further to discuss the possible effects on data movement either from or to multiple hosts. We extend the idea by examining conditions in a more complex situation where there are many platforms connected to each other and various data placement jobs between them are needed to be scheduled for different computing steps of applications.

The rest of the chapter is organized as follows. In Section 2.1, we present possible problems in the current large scale applications and give some motivating remarks. In Section 2.2, we focus on the data placement challenge and, in Section 2.3, we mention parameters that need to be used during scheduling to optimize data transfer. In Section 2.4, we explain data flow of large scale applications and present a data-aware system model. In Section 2.5, we explain data placement methodologies. And, in Section 2.6, we study approaches to integrate data scheduler with higher-level planners and other middleware tools.

2.1 Defining Data-Aware Distributed Computing

We define *data* as an illustration of information and concepts in a formally organized way to be interpreted and processed in order to accomplish computing tasks. Therefore, computing itself cannot be targeted as the only dilemma without providing necessary protocols to deal with storing and transferring information.

Rapid progress in distributed computing and Grid technology have provided collaborative studies and essential compute-power for science, and made data administration a critical challenge in this area. Scientific applications have become more data intensive like business applications; moreover, data management happens to be more demanding than computational requirements in terms of needed resources [64, 52].

Computation in science focus on many areas such as astronomy, biology, climatology, high-energy physics and nanotechnology. Although, applications from different disciplines have different characteristics; their requirements fall into similar fundamental categories in terms of data management. Workflow management, metadata description, efficient access and data movement between distributed storage resources, and visualization are some of the necessities for applications using simulations, experiments, or other forms of information to generate and process the data.

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 [29]. The LSST (Large Synoptic Survey Telescope) is scanning the sky for transient objects and producing more than ten terabytes of data per simulation [106]. 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 [26], processing of petabytes of experimental data remains the main problem affecting quality of the real-time decision making. In [78], data handling issues of Blast [37], a bioinformatics application, and CMS [1], a high energy physics application, have been discussed. Disadvantages of current methodologies to move data between execution processes and possible benefits by using a data scheduler have been explained.

Although data placement has a great effect on the overall performance of an application, it has been considered as a side effect of computation and either embedded inside the computation task or managed by simple scripts. Data placement should be efficient and reliable and also it should be planned cautiously in the process flow. We need to consider data placement and data scheduling as a significant part of the distributed structure to be able to get the best performance.

There are also numerous data-intensive applications in business which have complex data workload. For example, Credit Card Fraud detection systems need to analyze every transaction passing through the transaction manager and detect fraud using some models and historical data to calculate an estimated score.

Historical data and delivery of previous information should be managed in such a way that servers calculating the score should access data effectively. Moreover, there are many financial institutions providing data mining techniques for brokerage and customer services.

In addition, medical institutions are also one of the sources using computational network resources for their business. Particularly, large image transfers and data streams between separate sites are major problems in this area. Moreover, oil companies or electronics design companies may have long term batch processes. Therefore, commercial applications have short and long computational jobs and they also have small transactions and large data movement requirements. Since we have complex workload characteristics in business, a solution to data scheduling and data placement problem will be benefited by business as well.

Storing the real-time data is already challenging in experimental applications. In addition, efficient data transfer and organization such as metadata systems are also crucial for simulation-based applications which are usually executed in batch mode. Furthermore, those large-scale challenges in science necessitate joint study between various multi-disciplinary teams using heterogeneous resources. Thus, new approaches in data distribution and networking are necessary to enable grid technologies in scientific as well as industrial applications.

One technical challenge in science is high capacity, fast storage systems. There has been considerable effort in business to provide required information systems for commercial application; however, organization of the data for querying in petabyte-scale databases still remains an open issue, where relational databases cannot fit properly. Previously, there has been some work on special storage tools and metadata models developed for specific scientific applications [95, 105]. But we still need generic standards to integrate distributed resources with an open access mechanism to data.

2.2 The Data Placement Challenge

There are usually several steps in the overall process flow of large scale applications. One important aspect in distributed systems is to manage the interaction between computing tasks. Communication between different steps can established by the delivered information between them. The *data* (i.e. the delivered information), which characterizes the dependencies and connections, should be moved to supply input for the next computing processes between different tasks serving for different purposes.

We define *data placement* as a coordinated movement of any information between related steps in the general workflow. Reorganizing and transferring the data between separate tasks in a single server, moving data to different storage systems, or transmitting results of the previous tasks to a remote site are all examples of data placement. Data placement is not limited to only information transfer over a network; however, one common use-case is to transfer data between servers in different domains using the wide-area network. Interestingly, even data transfer inside a single machine is very important and may become a major bottleneck if not handled proporely.

We concentrate on several use cases of data placement tasks (as shown in Figure 2.6) and define important attributes that will affect overall performance and throughput of data transfer. Figure 2.6.a represents the data transfer in a single execution site where there is no network interference. Figure 2.6.b corresponds to the data placement between two different separate sites where data transfer between storage elements are performed over a network. Figure 2.6.c and Figure 2.6.d show data placement from or to a single storage element. Figure 2.6.e illustrates a more complex scenario such that various data placement tasks need to be scheduled between multiple hosts. Each scenario comes with a different set of parameters to be used during the decision making process of data placement execution.

2.3 Use Cases of Data Placement

First, we need to define attributes having a part in data placement and investigate their affects during the transfer operation in terms of efficiency and performance. Moreover, every parameter can be categorized according to the level in which it can be used such that some attributes need to be set automatically by the middleware and some of them can be defined by users. There are also some on-going projects preparing APIs for applications in widely distributed systems [34, 65]. Therefore, a detailed study describing possible use cases and defining factors in data placement is mandatory in order to propose a data scheduling framework.

2.3.1 Data Transfer in a Single Host

Processes running in a single machine might have special input formats necessitating former data to be converted properly and kept in a different storage space to accomplish desired performance. Moreover, we may

need to move some information to another storage device due to space limitations or storage management constraints. Different features of storage locations like availability and reliability, access rate, and expected quality of service to get information enforce reorganization of data objects. Therefore, some data files may be transfered to another disk storage in a single host due to administrative conditions, performance implications (like different block sizes in file systems), and requirements of running executables (such as formating and post-processing).

Movement of whole output or partial information in a single domain also has difficulty in terms of efficiency and performance. Besides, while copying a single file between two disks, we should deal with very common issues like space reservation, performance requirement, and CPU utilization. Load of the server, available disk space, file system block size, and protocols used for accessing low-level I/O devices are some factors for this simple example.

In current enterprise systems, we usually have specialized servers for different purposes like application, database, and backup servers. The most recent information is kept in fast servers and storage devices; then, historical data or other log information is transfered to other systems such as data warehouse systems.

Backup operations such as transferring data from multiple storage disks to fast magnetic tape devices requires multiplexing techniques to maximize the utilization and to complete the backup at minimum time. Several blocks from different files are read at the same time and pushed to the backup device in order to parallelize the operation.

The problem of data transfer within a single host seems simple. But actually it involves many sophisticated copy and backup tools in order to improve the efficiency. I/O scheduling in current operating systems with multiple storage devices is a good example of data placement in a single system. The OS kernel schedules data placement tasks in a single host by ordering data operations and managing cache usage. We usually have several channels, sometimes with different bandwidth, to access disk devices and usually more than one processor need to be selected to execute an I/O operation in a multiprocessor system. In addition to this, fairness between multiple I/O requests, avoidance of process starvation, accomplishment of real time requirements and resolving deadlock conditions are some other functions of the operating-system kernel in a single system. Consequently, complex queuing techniques for fairness and load balancing between I/O operations, and greedy algorithms for managing cache usage and disk devices (with different access latency

and capacity), have been developed to schedule I/O operations and to resolve the data placement problem in a single host.

Tuning the buffer size, using memory cache, partitioning data, using different channels for multiplexing are some factors. Data placement in a network also deals with the same concerns that we face in a single host, since in that case we also need to access file system layer and physical devices.

However, each of those factors and parameters affecting overall throughput and performance has different impacts in different degrees. We need to first search for the factors that have highest impact on performance, and then, try to optimize those key parameters. For example, a backup operation to a slow tape device from fast disk devices would not need cache optimization or parallel streams since read/write overhead of this slow tape device would be the bottleneck in the overall process. So, in such a case we do not need to optimize these parameters.

We identify some important factors for data placement in a single host as follows:

- server load, CPU and memory utilization,
- available storage space, and space reservation,
- multiplexing and partitioning techniques,
- buffer size and cache usage,
- file system performance (e.g., block size, etc.),
- protocol and device performance,
- I/O subsystem and scheduling.

Although I/O operations in a single server has many parameters influencing data placement performance and data transfer scheduling, we only focus on server side attributes like system load and utilization, space availability, and storage reservation to make the data placement models simpler in a single host.

2.3.2 Data Placement between a Pair of Hosts

Data placement between two hosts has many use cases such as downloading a file or uploading some data sets to be used by another application. Processed data in a single step in a large scale application with multiple, separate operational sites could be transferred to other related tasks to continue the operation in the overall workflow.

Some factors affecting the efficient data movement are changing network conditions, heterogeneous operating systems and working environments, failures, available resources, and administrative decisions.

Since network is thought as the main bottleneck in wide-area networks while sending large amount of data between a client and a server, increasing the number of simultaneous TCP connections is one method to gain performance. The *data-flow* process also includes streaming, and transferring of partial data object and replicas; however, a simple architecture build to transfer input/output files still remains as a major concern.

Some file transfer tools like GridFTP [62, 30] support multiple TCP streams and configurable buffer sizes. On the other hand, it has been stated that multiple streams might not work well due to TCP congestion-control mechanism [56, 57, 53]. Many simultaneous connections between two hosts may result in poor response times; and, it has been suggested that maximum number of simultaneous TCP streams to a single server should be one or two, especially under congested environments [57, 36].

Use of multiple connections is declared as a TCP-unfriendly activity, but outstanding performance results were obtained by employing simultaneous streams in the recent experiments [75, 79, 77]. Therefore, number of simultaneous TCP streams is one of the critical factors affecting performance for the transfers over a network.

Sudden performance decreases will be expected if number of concurrent connections goes beyond the limit of server capacity. The number of maximum allowable concurrent connections depends on both network and server characteristics, and selecting an appropriate value for each channel that will optimize the transfer in long term is a challenging issue in networked environments. Moreover, some protocols which are adjusted to utilize the maximum network channel bandwidth may have performance problems if more than one channel is used [73].

There are also various specialized TCP protocols [4, 28, 20] which change TCP features for large data transfers. TCP window buffer size may be the most important parameter to be tuned in order to decrease the latency for an efficient transfer. There have been numerous studies for fast file downloading [87]; and, many tools have been developed to measure the network metrics [27, 10] to tune TCP by setting the window size for optimum performance.

Bandwidth is the first metric that we look at to get an estimate about the duration of a transfer, but efficient use of protocols and network drivers can also serve as an important metric as they cause significant

improvement in terms of latency and utilization. Characteristics of the communication structure determine which action should be taken while tuning the data transfer. Local area networks and Internet have different topologies, so they demonstrate diverse features in terms of congestion, failure rate, and latency. In addition, dedicated channels such as fiber-optic networks requires special management techniques [20].

We also have high-speed network structures with channel reservation strategies for fast and reliable communication. Congestion is not a concern in such a previously reserved channel; and, transfer protocol should be designed to send as much data as possible for maximum utilization. We need to provide adaptive approaches to understand the underlying network layer and to optimize data movement accordingly.

Application and storage layers should feed the network with enough data in order to obtain high throughput. Network optimization may not have the expected effect if we cannot get adequate response from storage device. Buffer size that determines the volume of data read from storage should fit into network buffer used in the transfer protocol. Besides, server capacity and performance also have crucial roles and they identify how much data can be sent or received in a given interval. Therefore, storage systems have been developed for efficient movement of data [73, 80] by providing caching mechanism and multi-threaded processing.

We identify new parameters, which should be considered if data placement is performed over a network, as follows:

- simultaneous TCP connections (parallel streams),
- TCP tuning (send/receive buffer size, TCP window size, MTU, etc.),
- data transmission protocol performance (GridFTP, ftp, etc.).

Consequently, parameters that can be used in data placement problem between a pair of hosts can be extended by including the attributes related to the network such as degree of parallelization with concurrent channels, bandwidth reservation in dedicated networks, and transfer protocol optimization like tuning TCP window size. On the other hand, we are also bound by the server performance and we should consider efficiency, performance, and space availability of servers to manage data placement between a pair of hosts.

2.3.3 Data Placement from Multiple Servers to a Single Server

Nowadays, scientific and industrial applications need to access terabytes of data and they need to work with distributed data sets. One reason to keep information in distributed data stores is the large amount of

required data. Another concern is reliability such that replicas of information are stored in different systems apart from each other. Data management also focuses on replication in order to ensure data proximity and reliability. Alternatively, accessing separated data sets at the same time provides performance via parallel communication.

Data transfer from multiple clients to a single host and from single client to multiple hosts have many diverse samples. Data objects may be stored on separate storage servers due to space limitations or load balancing. Besides, running application modules may require remote access to get information generated by remote sites. Thus, a process scheduled in an execution site may need to start multiple data transfer jobs to obtain its input. Similary, output of in an execution site may need to be brought to multiple data sites.

One simple example is downloading multiple files or data objects from several sites in which we want to finish the operation in the minimum amount of time. In addition, we may prefer to move same data objects from different sites to ensure data availability in case of hardware and system failures in remote machines. Often, transferring files one by one is not preferred unless there is a special condition. Starting every download at the same time to get maximum utilization may seem as a good idea; however, efficiency of the data transfer is limited to server and network capacity.

Parallelization in network has two methods; one is parallel connections discussed in previous subsection, and the other is concurrent channels to every server. Parallel connections is obtained by opening multiple TCP streams between a pair of server. On the other hand, concurrent channels are maintained by multiple threads serving each data transfer from different sites.

The number of parallel connections from a client to each server is increased in order to get the maximum network utilization. Parallelization may have side effects similar to concurrency if generated load is over the limits of system ability. Both network properties such as bandwidth, failure rate, congestion, and server parameters like CPU load, memory and communication protocol have influences on parallelization. Therefore, number of parallel connections to a server, which is set according to the condition of network environment, is one of the critical parameters we should concentrate on.

Another example is to upload multiple files from different file servers to the execution site where we are running various tasks such that each task is using separate file sets staged to the execution site. Each task has diverse dependencies to some data objects stored in remote clients. We would like to optimize the

transfer so that maximum number of tasks can start execution. To simplify the problem, we can modify the objective and define it as maximizing the number of files arrived in the minimum amount of time.

One simple method for the given example is to deliver files with small size over high bandwidth connections to obtain the maximum number of transferred files. In order to get the best throughput over network, selecting large files and closer storage servers is another approach for optimization. In either case, we necessitate an ordering of data placement tasks. Even though if we are delivering replicas of a single object distributed across hosts, we still need to have a decision to minimize the required time.

Each connection from a server to the client that performs data movement operation might have different characteristics. Bandwidth, latency and number of hops a network packet should go through can be different. Thus, data placement jobs need to be scheduled according to conditions in network and servers.

Moreover, we expect different characteristics between uploading multiple files to a single server and downloading a file to multiple clients. Both cases look similar but they differentiate in terms of resources they use in the communication protocol. In the first case, transfer protocol should manage all channels and write data into the storage. In the second case, data should be read and pushed to all channels connected to the server. Read and write overhead in the transfer protocol affect operating cost in different levels so that specific actions are required to exploit the data placement for best performance.

We mainly focus on two additional factors in data placement from multiple servers to a single host;

- concurrent network connections,
- network bandwidth and latency.

Data placement from multiple servers to a single server is affected both by server and network conditions. We also need to consider the bandwidth and latency of the network to come up with a decision and order transfer tasks accordingly in order to obtain higher throughput. Furthermore, parallel connections to a server is a fundamental technique used to gain more utilization in the communication channel during the transfer operation.

2.3.4 Data Placement between Distributed Servers

There have been many 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 [55]. TCP buffer, window size,

MTU are some of the attributes used in the overall network transfer optimization. Moreover, high speed networks require different strategies to gain best performance. Parallel and concurrent transfer streams are used for high performance but unexpected results may occur in highly loaded networks [55].

Data servers are distributed on different locations and available network is usually shared like 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 [35, 14].

In widely distributed network environments, resources such as network bandwidth and computing power are shared by many jobs. Since data placement between multiple clients to a single host requires ordering in the data placement jobs, data transfer between distributed servers is even more complicated. We necessitate a central scheduling mechanism which can collect information from separate sites and schedule data placement jobs such that maximum throughput in minimum time is achieved.

In more complex realistic situations, the network is jointly used by several users unless dedicated channels are allocated. It is really hard to obtain the state of network condition especially in widely distributed system.

Parameters like bandwidth and latency that need to be used during the decision process are dynamically changing. We can measure the values of attributes like CPU load, memory usage, 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.

In order to clarify the data placement scenario we state a simple example in which multiple files in distributed data centers are need to be transfered to a central location and we need to upload results of previous computations to those data centers. A job which requires data sets from other data objects has been scheduled and started execution in a computing site, so we need to transfer data sets from data servers to the execution site. We may have some other tasks in the execution site trying to send files to this data servers at the same time. We may also need to transfer data to the data server from some other sites located separately.

Since each data movement job is competing with each other, a decision making is required to have them ordered and run concurrently. Without a central scheduling mechanism, saturation both in network and servers is inevitable. 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 placement tasks. On the other hand, decision on the number of parallel streams and concurrent connections has become more difficult such that there may be more than one task transferring data at the same time.

As a conclusion, the problem itself is complicated and it depends on many parameters. Figure 2.1 summarizes some parameters discussed in this section. We analyze the problem starting from simplest scenario and extend to more complex situations by stating critical key features affecting the performance. Besides, scheduler should be able to make rapid choices and dynamically adapt itself to changing situations. Instead of applying composite algorithms, simple scheduling techniques will be efficient to have at least satisfactory performance results.

	In Single Host	Between a Pair of Hosts	Multiple Servers to Single Server	Between Distributed Servers
Available Storage Space	✓	✓	✓	✓
CPU Load and Memory Usage	✓	✓	✓	✓
Transfer Protocol Performance		✓	✓	✓
Number of Parallel Connections		✓	✓	✓
Network Bandwidth and Latency			✓	✓
Number of Concurrent Operations			✓	✓
Ordering of Data Placement Tasks				✓

Figure 2.1: Key Attributes affecting Data Placement

2.4 Data Flow in Large Scale Applications

A simple architectural configuration of distributed computing for large scale applications can be evaluated in four phases. First, we require workflow managers to define the dependencies of execution sequences in the application layer. Second, higher level planners are used to select and match appropriate resources. Then, a data-aware scheduler is required to organize requests and schedule them not only considering computing resources but also data movement issues. Finally, we have data placement modules and traditional CPU

schedulers to serve upper layers and complete job execution or data transfer. Figure 2.2 represents a detailed view of data-intensive system structure.

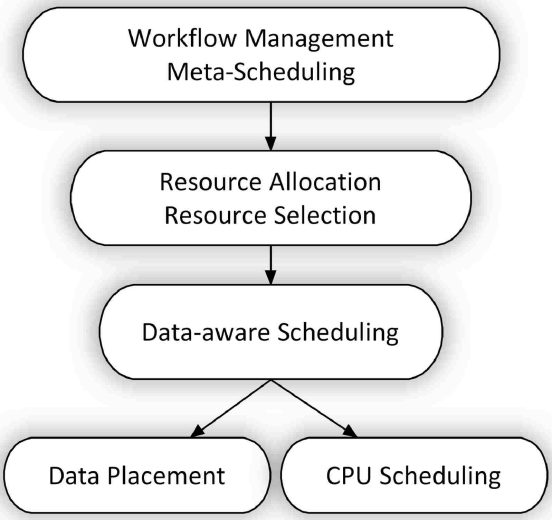


Figure 2.2: Data-aware System Model

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. We usually transfer data to separate centers for individual analysis; even though, different science groups may require only some parts of the data such that data set groups from different activities may be used to extract some features. A simple flow in the overall process is shown in Figure 2.3.

Execution sequence and data flow in scientific and commercial applications need to be parallel and should provide load-balancing in order to handle complex data flows between heterogeneous distributed systems. There have been recent studies for workflow management [83, 100, 101], 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. Therefore, data placement is an effective factor in workflow management, and scheduling of data placement tasks need to be integrated with workflow managers.

Transfers especially over wide-area 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 [59, 103, 43]. Replica catalogs, storage management and data movement tools are addressing some of those issues in the middleware technology

[62, 84, 51]. Importance of data scheduling is emphasized by [78], and data scheduling has been stated as a first class citizen in the overall structure.

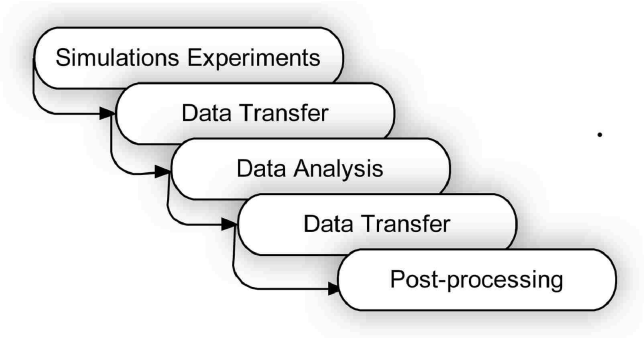


Figure 2.3: Data Flow in Large Scale Applications

Traditional CPU-based scheduling systems do not meet the requirements of the next generation science and business applications in which data sources should be discovered and accessed in a flexible and efficient manner during the job execution. Moreover, developments in distributed computing infrastructure such as fast connections between widely separated organizations and implementation of huge computational resources all over the world made data scheduling techniques to be considered as one of the most important issues. Data intensive characteristics of current applications brought a new concept of data-aware scheduling in distributed scientific computation.

Simulating the Data Grid to investigate the behavior of various allocation approaches have been studied [94, 15, 82]. Beside the simple greedy scheduling techniques such as Least Frequently Used and Least Recently Used, there are also some economical models handling data management and task allocation as a whole while making the scheduling decision [107]. Another recent work concludes that allocating resources closest to the data required gives the best scheduling strategy [92, 93]. 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.

2.5 Data Placement Methodologies

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 trans-

parency and data migration. Data placement in distributed infrastructure should use parallelism and concurrency for performance issues. 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.

Data placement jobs have been categorized in different types (transfer, allocate, release, remove, locate, register, unregister) and it has been stated that each category has different importance and optimization characteristics [78]. In order to simplify the problem, we focus only on data movement strategies in data-intensive applications. One important difficulty 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 store 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. However, there may be some external access to the storage preventing the data scheduler to make good predictions before starting the transfer. Allocating space and ordering transfers of data objects are basic tasks of a data scheduler. Different techniques such as smallest fit, best fit, and largest fit have been studied in [79]. Data servers have limited storage space and limited computing capacity. In order to obtain best transfer throughput, underlying layers in server site that may affect the data movement should be investigated cautiously. Performances of the file systems, network protocol, concurrent and parallel transfers are some examples influencing data placement efficiency.

Another important feature of a data placement system 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 acknowledgement. A data transfer model should minimize the possibility of failures and it should also handle faulty transfer in a transparent way.

We necessitate a scheduling mechanism which can collect information from separate sites and schedule data placement jobs such that maximum throughput in minimum time is achieved. Such that, each data movement job is competing with each other; therefore, a decision making is required to have them ordered and run concurrently [42]. Therefore, the data scheduler plays an important role in order to enhance the overall performance. We need to have control on operating data transfer operations not only to come up with better tune-up predictions but also to supply better ordering to avoid starvation and contention [42]. Without

the help of a central management, all transfer jobs submitted in the system can operate simultaneously and lead to contention with opening so many parallel network connections and saturation with creating so much load on certain resources.

Since many other data transfer jobs can share the common resources and we need to schedule transfer jobs and set tune-up parameters before initiating the data transmit operations, we propose a virtual connection layer inside the data scheduler to keep track of network statistics. The data scheduler opens a virtual channel for every source-destination pair encountered so far, and makes prediction for the data transfer over this communication pattern. The concept of virtual communication channel defined as an abstract level inside the data scheduler, enables us to utilize network predictions in the overall system. Moreover, attributes affecting the predicted values can be fetched and virtual channel capabilities are updated dynamically. The network parameter prediction is a sub-process besides the main central data placement manager serving for better scheduling decisions. Figure 2.4 represent the overall structure of a data scheduler using network prediction values.

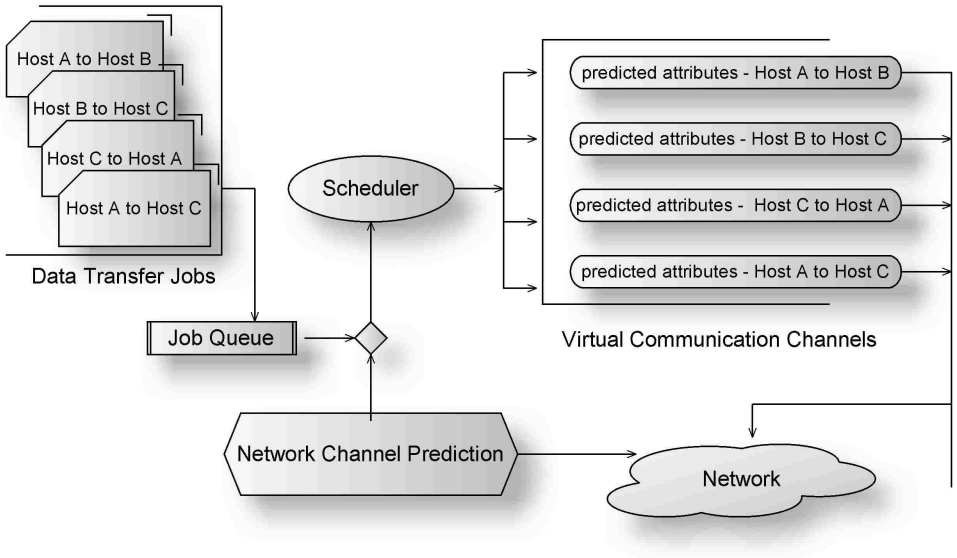


Figure 2.4: Virtual Communications Channels for Paramater Setting

2.6 Integration with Higher-level Planners

One important issue in distributed systems is to decide on best data access strategy. An application can use remote I/O or data staging in which data files are transferred to the execution host. As it has been explained in [99], that decision process depends both on the characteristics of the submitted application and also the current condition in the environment. Choosing between remote I/O versus staging in large scale distributed applications falls into two layers in the data-aware distributed system structure. In the meta-scheduling phase, where we generate the workflow for the application and select appropriate resources, we gather information specific to the application like task dependencies and frequency of accessing the same portion of data. In the following phase, where we perform distributed scheduling concerning computational requirements and data requirements of the application, we collect information specific to the environment like latency, bandwidth, and local storage performance.

Staging application data is the most common approach. On the other hand, accessing data using remote I/O protocols brings some advantages both in application layer and in the implementation of the middleware. The data scheduler is responsible for organizing submitted application tasks according to available resources in the environment. Besides computational resources required to accomplish the given task, we also deal with managing input and output data sets. During the decision process, data placement plays an important role such that scheduler is supposed to interact with lower level planners and also data resource manager to select the best possible resources.

In case of using the staging approach; first, higher level planners communicate with data placement scheduler to stage-in data to the execution site. Then, traditional CPU scheduler is used to execute the application; and later, data placement scheduler performs staging of the output data. In the overall data-intensive system model, the decision of selecting remote I/O or data staging method is accomplished before generating the workflow of the distributed application.

The method proposed in [99] for choosing a data access technique, remote I/O or staging, for large-scale distributed applications, is used in the planning phase before submitting tasks to the next level resource brokers. Though we need to gather information about the environment and we need to be aware about the input data requirements of the application before making any decision about the data access pattern, the

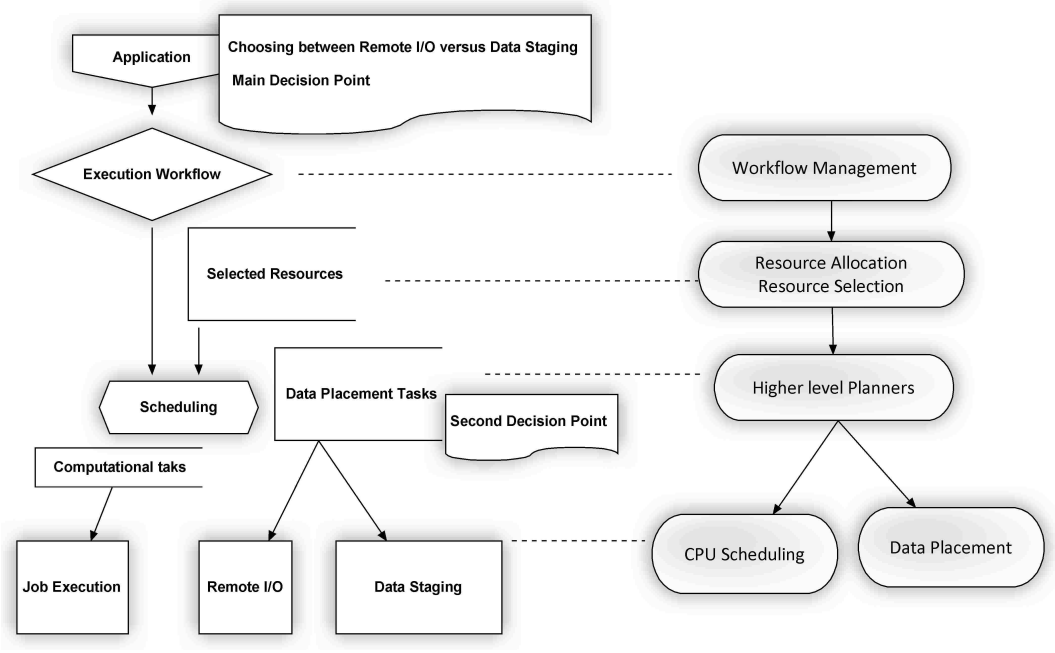


Figure 2.5: Decision Points: Remote I/O versus Data Staging

model used to choose remote I/O versus data staging can also be used conditionally in the scheduling phase. Figure 2.5 gives a broader view of the proposed model for large scale applications in widely distributed environments.

The data placement scheduler is supposed to deal with the setup and preparation of the environment even if remote I/O technique is used in the running application. Orchestrating data-related operations is the main task of a data-aware scheduler. The data-aware scheduler should be capable of making decision according to the parameters that are dynamically changing in distributed environments. The second decision point for choosing remote I/O versus data staging is in the scheduling phase. On the other hand, modifying the data access pattern of an application in the execution flow is limited with the supported data access protocols in the system. If data staging can not be accomplished, the scheduler can modify the data placement tasks without affecting given workflow such that access to data files in the execution of the application are replaced with remote I/O calls. The application access input data using remote I/O protocols and write output back over a remote storage server. In order to switch data access pattern transparently from staging to remote I/O, file accesses should be virtualized. Fortunately, recent studies and file system virtualization tools like Fuse [5] and Parrot [16] enable us to have a second decision point in the structure.

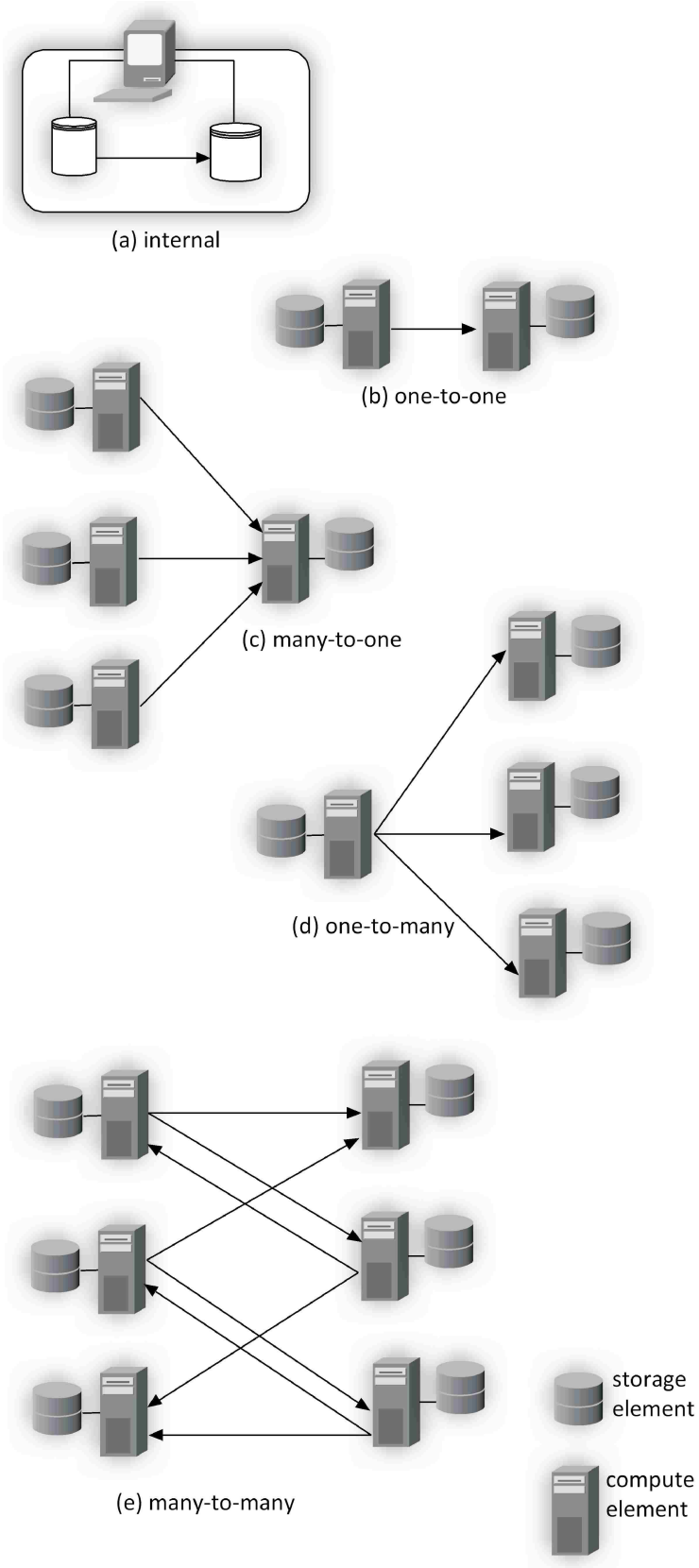


Figure 2.6: Data Placement Scenarios

Chapter 3

Lessons Learned from Computer Architecture

In the recent past, microprocessors have been one of the fastest growing technologies. In the early models, accessing data was one of the major problems and it is still a crucial issue today; such that, 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. In order to overcome the speed gap between arithmetic operations and memory access operations, many techniques have been applied. Scheduling the execution of instructions starts with out-of-order execution technique used in early x86 designs and extends to dynamic scheduling algorithms in which operations are buffered in the issue phase before they are sent to execution units. Moreover, pre-fetch registers, multi-level caches, complex branch prediction algorithms are some of the methodologies used in processor design. Execution stages are split into execution units and also into multiple pipelines in order to enhance performance using pipelining and superscalar design. Load-store operations are separated from other execution units and I/O instructions are scheduled and executed by specialized units in microprocessors.

In this chapter, we focus on problems in accessing and storing data that have been encountered in the early phases of processor design, and also analyze how those issues have been resolved to enable today's fast and efficient microprocessors. We study how similar challenges can be addressed in distributed systems. Since small and large scale systems have similar problems, we can utilize approaches in microprocessor and operating system kernel architectures to come up with a broader perspective in which we can extend known methodologies to be used in distributed systems.

In Section 3.1, we discuss basic computer architecture. In Section 3.2, we explain the distributed data management strategies and analyze possible solutions for similar problems learned from microprocessor architecture. Then, in Section 3.3, we present a data-aware distributed computing paradigm underlining the fact that different layers of computing systems have similarities in data and CPU management subsystems. In Appendix A.1, we explain microprocessor evolution. A brief summary explaining how microprocessors evolved and how processor technologies developed have been given. In Appendix A.2, limitations and difficulties affecting microprocessor design, and challenges in terms of data management have been mentioned.

In Appendix A.3, microprocessor architecture has been discussed and basic techniques such as caching, pipelined and superscalar execution, and also progress of those methodologies throughout the history of microprocessors have been explained.

3.1 Computer Architecture Overview

The code stream is an ordered sequence of operations and commands that tell the computer what to execute [98]. In modern microprocessors, instructions have been mainly classified into two categories as arithmetic instructions and memory-access instructions [58]. Memory-access instructions command the parts of the processors, usually called memory-access hardware, that deal with movement of data from and to the main memory such as load and store. Arithmetic instructions are used to perform arithmetic calculations and logical operations which are dealt with a specialized hardware, generally called ALU. On the other hand, branch instructions, a third type of instruction, are special type of memory-access instructions that only access code sequence instead of data storage [98]. Memory-access instructions are significant because access to the main memory is one of the crucial challenges in microprocessor design in terms of performance.

All computers consist of three basic structures; the storage to read and write input and output data, the arithmetic logic unit for computing, or simply modifying the input data; and the bus to transmit data between ALU and storage [67]. Moreover, a computer model can simply be defined as sequences of read, modify, and write operations [98]. Instructions are fetched over instruction bus from storage area to the ALU; then, input data is transmitted to the input port of the ALU, and modified data is stored to the storage device over the data bus. Since input data should be transmitted to the computing unit before performing any operation, register files have been used starting in early computer designs to place the data storage close to the ALU so operands can be accessed instantaneously [98]. The basic role of memory-access hardware is to transmit data between the register file inside the processor and the main memory.

3.2 Data Management in Distributed Systems

We approach the data scheduling problem in a different way and analyze data access and data management issues in different layers of computer systems starting from microprocessor level, extending to operating system level, and then, investigating data handling in distributed systems. We study problems and challenges

in data access and data storage, and we investigate applied techniques in small and large scale systems. Studying and analyzing in different scales will enable us to better understand the data management problem in distributed systems.

Two basic components of a distributed system are computing and storage resources. Data which may be generated by simulations or collected from scientific experiments is stored in particular sites so it can be accessed by applications and shared between computational resources. Since we have huge data to keep and also we need to control the unity of data, specialized data servers and data storage sites have been formed. Size of data is usually very large and cannot be kept in every system; and synchronizing to a central data site in which other elements can utilize needed part of data is much easier than distributing into every component in the general model.

Moreover, data is usually stored in a hierarchical manner to provide fast access to requested information. A storage system contains hardware components such as tape libraries and disk arrays, management services to organize and storage data, and also networking to provide other elements access to information in a distributed environment. Figure 3.1 gives an overall view of distributed system architecture.

On the other hand, computational resources, cluster of many servers to afford high computing power, are distributed and connected to each other over a network. Network service is one of the most important parts in a distributed environment since it makes separated elements reachable to each other. Besides, data which is not locally stored should be accessed in an efficient and transparent manner. Therefore, necessitated input data is either transferred and stored in a temporary space or accessed remotely over the network. Due to the nature of interconnects between distributed elements there are many factors affecting performance such as latency and bandwidth. This leads to staging and caching services in order to make data easily and quickly accessible.

3.3 Generic Data-aware Computing

Microprocessors have been mainly classified into two categories as arithmetic instructions and memory-access instructions. We also classify operations in the distributed systems into two categories as computational tasks and data access/movement tasks. Data movement tasks are responsible for bringing the input data and carrying back the output data. Data is moved from and to the execution site such as stage-in and

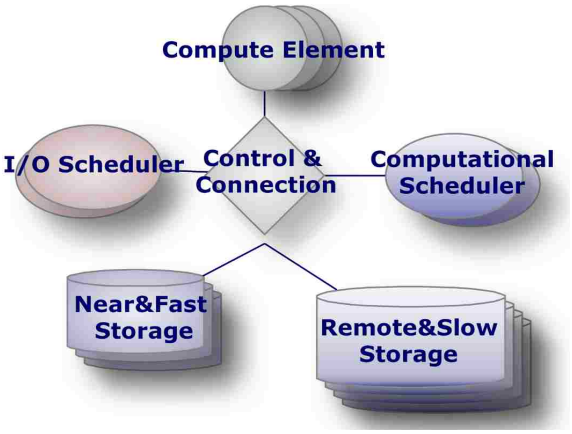


Figure 3.1: Generic Model

stage-out operations. Computational tasks are tied to each other by input/output patterns, and staging operations define the data dependencies in the workflow such that staging data in and out plays an important role in the overall performance.

In distributed systems, we have computational nodes, data storage elements, and network interconnects to transmit data between computational nodes and storage elements. In microprocessors, an instruction starts in the fetch phase, moves to a decode phase, and it is sent to the execution phase, then to the write phase [63, 98]. In distributed systems, different service layers have been defined and overall load is shared between separate components. Therefore, there are different software and hardware elements serving for specific purposes to maintain the overall system. Basic components of a distributed system are computing and storage resources which are reachable to each other over network connections. Specialized data servers and data storage sites have been formed due to storage requirements of today's applications. Figure 3.2 gives an overall representation.

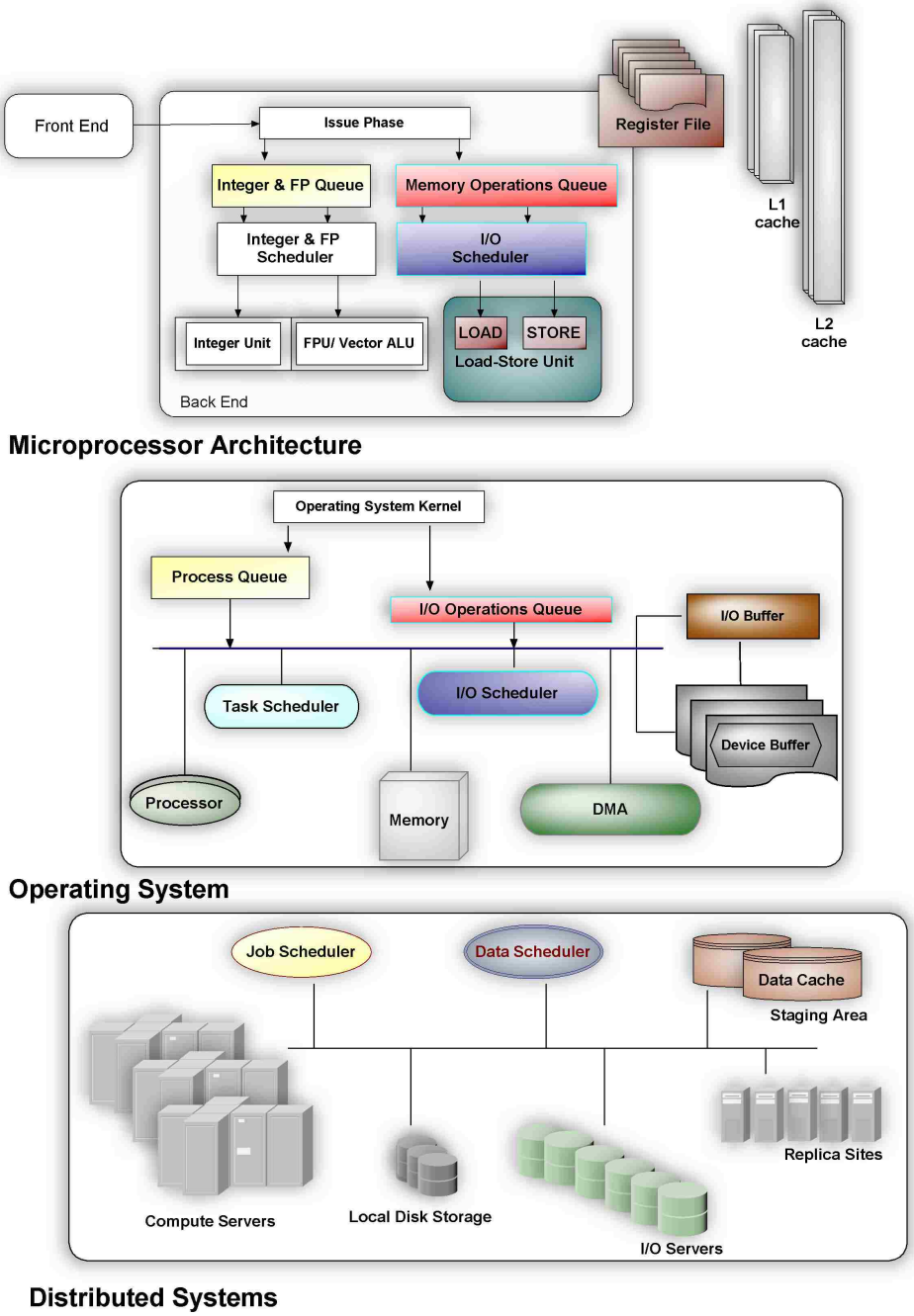


Figure 3.2: Generic Data Management

Chapter 4
Adaptive Scheduling

We have studied data access and data management in several layers of computer organization. As it has been explained in Chapter 3, one major bottleneck is the latency of the communication link such that the memory wall arise from limitations between interconnects. In this chapter, we investigate the aggregation of data placement tasks as it has been done in microprocessor design by using intelligent units for prefetching instructions and caching data .

In our data-aware system model, the data placement scheduler is responsible for orchestrating the management and coordination of data transfer jobs. We have data transfer jobs being submitted into the scheduler as a sub-part of a broader set of processes in an execution workflow. The first goal of the data placement scheduler is to minimize the transfer time since other components in the workflow might depend on this data placement job such that they might be waiting for the data to be ready. On the other hand, we also need to consider other jobs waiting in the queue or jobs currently being executed. Therefore, we tune-up system resources and order data placement jobs to minimize the completion time of every single job while trying to maximize the overall throughput.

In this chapter, we first focus on the level of parallelism in two stages of data transfer scheduling: (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 to better utilize system resources. According to previous studies and our first hand experiences [112], there is a threshold for both the number of concurrent jobs and the number of parallel streams for a single job. Such that, we see no performance gain after the threshold, even though there might be performance degradation due to resource starvation.

One important parameter affecting the performance of a single data transfer is *the number of parallel data streams*. We use multiple data streams for fetching data in order to fully utilize the network bandwidth. Since we are limited by the latency in the network, the buffer size optimization and parallelism have great impact on minimizing the overall data transfer time. Besides, we start multiple operations for better utilization of the system resources. Other processes sharing the same system resources like CPU, disk, network

have influence also on the overall performance. Excessive use of these resources may lead to some problems like resource starvation. In those cases, we need to adjust the level of parallelism to avoid resource starvation.

One approach is to measure the network and system statistics and come up with a good estimation for the parallelism level. However, those techniques require extensive measurements and usage of historical data. Besides, the estimated value might not reflect the best possible case due to the dynamic characteristics of the distributed environment. We have designed an alternative approach in which instead of making external measurements over the network, we make use of the information gathered from the data transfer operations that are currently active. We propose an adaptive approach in which we measure the throughput of every operation inside the scheduler and gradually increase the level of parallelism while transfers are in progress.

Number of concurrent jobs running at the same time is increased gradually if there is any performance gain for the overall throughput. We also adapt the number of parallel streams in single transfer according to network conditions. We start with a few streams and increase the number of streams gradually up to the point where no more performance impact is observed.

4.1 Level of Parallelism in Data Transfers

The possible parameters that are required to accomplish a data transfer job are : source URL, destination URL, file size, user certification, data transfer protocol, and the number of parallel streams. We basically focus on modern data transfer protocols, so that certification is used to authenticate a user, and then we are able to open multiple streams to transfer data in parallel. The characteristics of the data transfer protocol have consequences on setting up parallelism and arrangement of the scheduling order. We ignore the complexity of data transfer protocols and some other user parameters, for simplicity. We only focus on several key arguments (like size of data, source and destination hosts) to define a data transfer job J as follows:

$J(\text{data transfer}) =< \text{source hosts}, \text{destination hosts}, \text{data size} >$

The data transfer scheduler accepts multiple jobs in a nondeterministic order, say: $< J1, J2, J3, \ldots >$. Completion time of a job also depends on environment conditions such as network and system load. In order to increase the throughput, we use multiple streams, so completion time T, depends on the number of parallel streams used for the transfer: $T(J, \text{parStreams}, \text{Environment})$

We give the control of setting the level of parallelism to the scheduler. We tune the number of parallel streams according to the environment conditions, and also taking into consideration the fact that there are other jobs currently running in the system. For each *source-destination pair*, we keep track of jobs with all possible parameters to be used in subsequent scheduling decision as follows:

In a virtual communication table, we store information about the running jobs that use the same source and destination hosts during the transfers. We also maintain a record about the current throughput value that has been seen in this source-destination pair. The number of parallel connections is used to the set parallelism level in a data transfer from this source A to destination B. Throughput value is calculated from the last successful single transfer operation and it reflects the current state of the system. We use the latest throughput value to make a decision about whether to increase the number of parallel streams and the number of concurrent jobs, using this source-destination pair.

We would like to emphasize the difference between the *Get* and *Put* operations during data transfers in terms of the load put on the host machines. We gather information about all ingoing data transfer operations (Put) into a host, and all outgoing data transfer operations (Get) vice versa.

$JL(source) =<$ total number of jobs, total number of connections, best throughput, up/down?, $reason >$

$JL(destination) =<$ total number of jobs, best throughput, $up/down?, reason >$

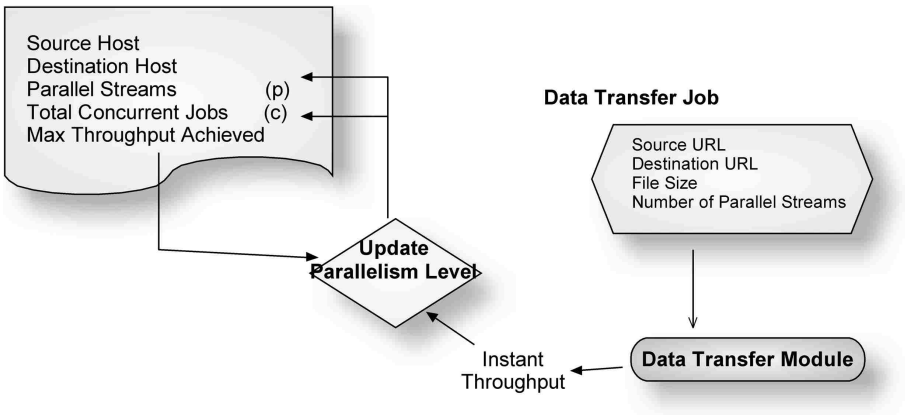


Figure 4.1: Adaptive Parameter Tuning

By the total number of concurrent jobs, we mean all the data transfer operations scheduled at the current time-period where those jobs are accessing and using the resources from source/destination host. We also maintain a record of whether this host is accessible or not. In case an error has occurred, we store the reason

for failure, and mark in the virtual communication table. If there is a temporary error (like, error in data transfer service, or host machine down for a while), we can recover and start using the previously saved setting in the table.

Figure 4.1 shows the overal structure in adaptive data scheduling. For simplicity, we reduce parameters affecting T into two key attributes: (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 find the best possible settings for c (concurrency level) and p (parallelism level) in an adaptive manner without using the external measurements form the network prediction tools. $T =< \mathrm{J}, \mathrm{c}, \mathrm{p} >$

4.2 Adaptive Data Transfers

One important issue in tuning wide-area TCP network is setting the buffer size to maintain network pipe full for high throughput. The optimal buffer size is related to bandwidth-delay product which is the product of the data link capacity (bandwidth of the bottleneck link) with end-to-end delay (the round trip time of the connection). However, setting optimal buffer size has been studied in data communications for various configurations ranging from Ad Hoc mobile networks to high performance data transfers. There have been many studies that investigate diagnosis techniques to measure bandwidth and round-trip time of a given connection [54]. The bandwidth-delay product can vary in a shared wide-area network environment. Thus, initial measurements and estimated buffer size values may not reflect the optimal value for the entire lifetime of a data transfer operation. Therefore, an adaptive methodology to set the buffer size on the file has been implemented in [61, 60]; such that, a dynamic auto-tuning approach in user-space has been proposed. Moreover, a lightweight technique is implemented to measure the bandwidth and end-to-end delay.

Instead of making extra measurements, bandwidth value is estimated using samples from actual transfers in currently progress. For round-trip time, small packets are sent over the control channel to measure end-to-end delay [47, 61]. In summary, there is no extra load in network for calculating bandwidth-delay product. And, buffer size has been tuned on the fly according to the changing conditions in the environment. Dynamic-right sizing [111, 47] has been applied to file transfer operations in Grid by extending the

GridFTP protocol [30, 62, 69]. In our work, we also implement adaptation by fair-sharing system and network resources. Instead of making measurements and using historical data to set level of concurrency and parallelism in data transfer scheduling, we calculate best estimate using the information from previous or current running data transfer operations. Thus, we do not pollute the network with extra probing packets and do not put extra load to the system by extraneous calculations for accurate parameter settings. Alternatively, we dynamically set level of parallelism, both number of concurrent jobs and number of parallel streams, by observing the achieved throughput for each transfer operations and gradually tuning parameters according to previous or current performance merit.

There are several studies for configuring the GridFTP control parameters like the number of parallel streams [69]. In order to optimize the level of parallelism, we need information such as packet loss rate, end-to-end delay and available bandwidth. Moreover many models have been studied in the literature to estimate accurate parameter configuration for parallelism [112, 69]. In one recent study, an automatic parameter configuration for GridFTP protocol has been studied [70]. The Grid-APT (Automatic Parallelism Tuning) explained in [71, 70], measures throughput for each chunk transferred and adaptively increase the number of parallel TCP connections if there is performance gain.

Since we need to reset the number of TCP data channels to be used in the transfer, there is an overhead in reconfiguring the GridFTP connection. In order to minimize the effect of this overhead for reconfiguring the GridFTP control parameter, large blocks called chunks are transferred at every data operation call [71]. Grid-APT calculates the goodput of each chunk using the chunk size and transfer time of that chunk. Then, a numerical computation method called Golden Section Search is applied [71]. The main objective is to only use information measureable in the Grid middleware while optimizing the number of parallel TCP connection for achieving high throughput. The numerical method used in [71], is suitable for searching a value that maximizes a convex function in a given range. We gradually increase the number of parallel streams and find the best parameter achieving the highest throughput. Moreover, some more complicated techniques for numerically searching best parameter have been examined in [70]. Additive increase and multiplicative decrease, multiplicative increase are some of them in which we first initialize the number of parallel TCP connections, then transfer a fixed chunk and measure the goodput and round-trip time. Later, if there is gain in terms of performance we increase the parallelism level, N, by a constant factor, $N < \alpha N$

[70, 71]. The upper limit for goodput value is defined as $(NW)/R$ where R is the round-trip time, N is the number of optimum parallel streams and W is the TCP buffer size. Initially in [71], the parallelism level is adjusted and increased whether the new goodput G_n has better results than the one G_{n-1} in previous step. The measured goodput for successive transfers of chunks will be limited with the upper bound that is for optimum parallelism level. If the number of parallel streams is larger than the optimal value, a decrease in goodput value will be seen such that we stop and terminate the algorithm when we reach a near optimum value for the parameter.

Using multiple parallel streams in data transfer is simply aggregation of TCP connections. Instead of a single connection at a time, multiple TCP streams are opened to a single data transfer service in the destination host. We gain larger bandwidth in TCP especially in a network with less packet loss rate. Parallel connections better utilize the TCP buffer available to the data transfer such that N connections might be N times faster than a single connection [70, 71]. Besides, aggregating TCP connection will speed up the slow-start phase in TCP [70]. Conversely, we observe a major throughput decreases due to overhead in TCP stack, if number of parallel streams, or aggregated TCP connections, are over the optimum value.

We have also implemented a similar technique as a special data transfer scheduler module. The adaptive transfer module starts with an initial best value for the number of parallel streams set by the scheduler. Then, it measures the transfer time of each chunk transferred and calculates the throughput. The dynamic feature of this GridFTP transfer module enables us to transfer data by chunks and also set control parameters on the fly. We keep the record of best throughput for the current parallelism level. The actual throughput value of the chunk transferred is calculated and the number of parallel streams is increased by one 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. The following section gives better explanation of the methodology with experimental results.

One recent work studies run-time adaptation of data placement jobs [76]. It is emphasized that we have a dynamic environment in the Grid and this work comes with a prototype to monitor environment conditions and to update the data placement scheduler required with information to tune up control parameters. The data placement scheduler uses tuned parameters for performing run-time adaptation. There is a tuning infrastructure in which environment information is provided by external profiler like memory, disk network

profilers. The parameter tuner executes periodically and requests information from profilers to come up with a decision which is to be used by the data transfer scheduler to tune up parameters in data transfer modules. Our approach differs from this structure by not depending on any external measurements or profiling results. Moreover, adaptability is embedded inside the scheduler such that each data transfer jobs returns with its performance information to be used by the scheduler to tune up following requests. Instead of probing the system in order to get profiling information, we just use performance outcomes from actual data transfers for parameter tuning. This approach also does not require any complex model for parameter tuning. The scheduler's decision adapts itself to the environment condition no matter what is really creating the major bottleneck. In summary, memory shortage or insufficient network latency might be the actual reason for not letting us to increase the number of parallel streams or the number of concurrent jobs. But, gradually improving parameter setting brings a near optimal value without the burden of finding the major bottleneck in data transfer.

Adaptive data placement has also been studied in various data management architechtures like in [108]. Beyond that, adaptive data transfer protocol are used in wireless network for better energy consumption [49]. Data transfer protocol adapts itself to the changing environment conditions such that data transferred rate is increased when communication parties are closer to reduce power consumption and data transfer rate is decreased when there is long distance between communication parties. Data transfer rate is adjusted using the estimated and measured values in each transfer period.

$D_{estimated}(t+1) = \alpha D_{measured}(t) + (1-\alpha) D_{estimated}(t)$

Similarly, we also try to utilize the underlying resources and fill the network pipe as much as possible in order to minimize completion time of a transfer. Therefore, we set maximum allowable parallelism level during data transfer scheduling.

4.3 Experiments in Parameter Tuning

In order to test our methodologies, we use the LONI [9] environment. Table 4.1 presents the network characteristics of our test system. First, we have performed experiments to see the effect of number of parallel streams in GridFTP transfers for small and large file sizes. As can be seen in Figure 4.3, we present average throughput results for data transfers with parallel TCP streams. Later, we have tested effect of

Table 4.1: Test Environment

source host = queenbee.loni.org		RRT (ms)		
	Destination	min	avg	max
Linux machines	eric.loni.org	0.541	0.548	0.555
	louie.loni.org	5.105	5.131	5.159
	oliver1.loni.org	2.438	2.456	2.466
	poseidon1.loni.org	5.32	5.334	5.343
IBM machines	ducky.loni.org	5.107	5.129	5.142
	neptune.loni.org	5.325	5.34	5.355
	zeke.loni.org	2.473	2.505	2.525
	bluedawg.loni.org	7.992	8.005	8.034

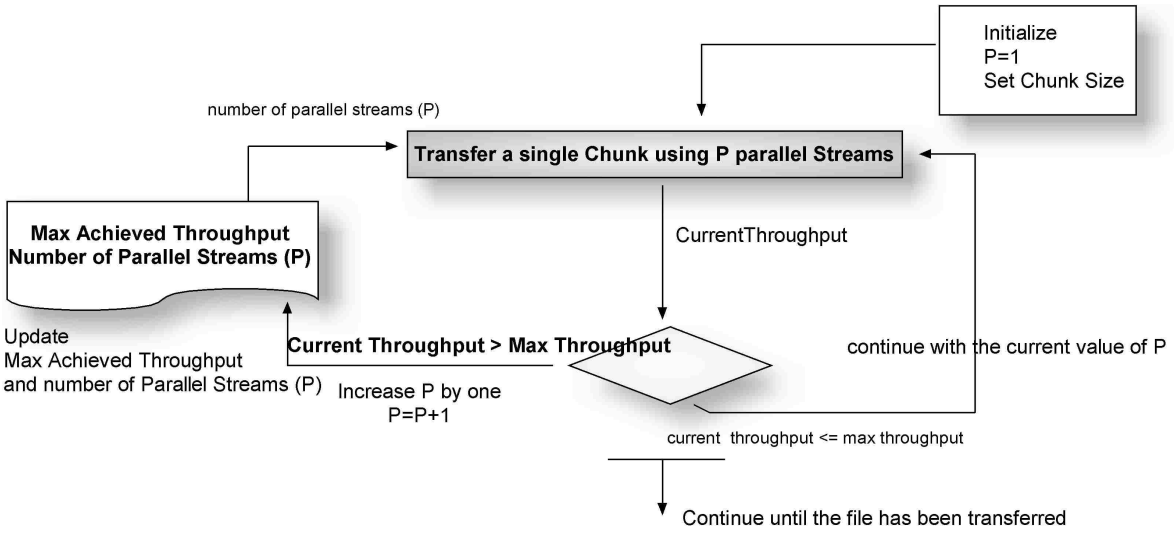


Figure 4.2: Setting the Parallelism Level

concurrent jobs performing transfer operations at the same time. Figure 4.3 presents average throughput of concurrent jobs running at the same time. Test results in experimenting the effect of concurrency level shows more inconsistent behaviour. According to our first hand experience, data transfer jobs hang and stop responding due to resource starvation usually after 25-30 concurrent jobs.

Figure 4.5 shows results from our adaptive transfer module in which we gradually increase the number of parallel streams. The adaptive setting of parallel streams make use of the best throughput results achieved so far and and adjust the parallelism level adaptively. As can be seen in Figure 4.6, it does not get affected by fluctuations in the network affecting instant throughput results. Figure 4.2 gives a glimpse of the algorithm used to implement the transfer module which set parallelism level adaptively.

We also study aggregation of requests in order to increase the throughput especially for transfers of

small data files. 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. The main performance gain comes from decreasing the amount of protocol usage and reducing the number of independent network connections. Thus, Stork makes better use of underlying infrastructure by coordinating and arranging data placement jobs.

Our test set includes 1024 transfer jobs from ducky to queenbee (rtt avg 5.129 ms) with a 5MB data file per job. Figure 4.7 shows performance measurements according to various parameters. 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.

Beyond that, we also show the effect of connection time in Table 4.2. Main advantage of aggregating data transfer jobs and combining them into a single job, is to elimine the cost of connection time for each seperate transfer. In a single operation we transfer multiple file over a single connection to the data transfer protocol.

Table 4.2: The Effect of Connection Time in Data Transfers
put operation (from queenbee.loni.org)

file size	number of files in a transfer	time(secs)	file size	number of files in a transfer	time (secs)
louie.loni.org10MB	1	1.59	eric.loni.org 10MB	1	0.39
	2	1.95		2	0.52
	3	3.39		3	0.67
	4	3.68		4	0.8
	5	4.71		5	0.9
	6	6.69		6	1.03
	7	5.94		7	1.62
	8	6.77		8	1.35
	9	7.4		9	1.7
	10	8.18		10	2.5
100MB	1	9.81	100MB	1	2.4
	2	17.82		2	2.65
	3	27.7		3	5.26
	4	32.93		4	7.87
	5	43.86		5	9.48
	6	49.46		6	11.47
	7	60.64		7	11.73
	8	64.48		8	11.35
	9	72.59		9	11.92
	10	80.5		10	13.39
1GB	1	79.74	1BG	1	17.46
	2	155.78		2	34.74
	3	228		3	54.2
	4	312.36		4	63.25
	5	381.89		5	78.82

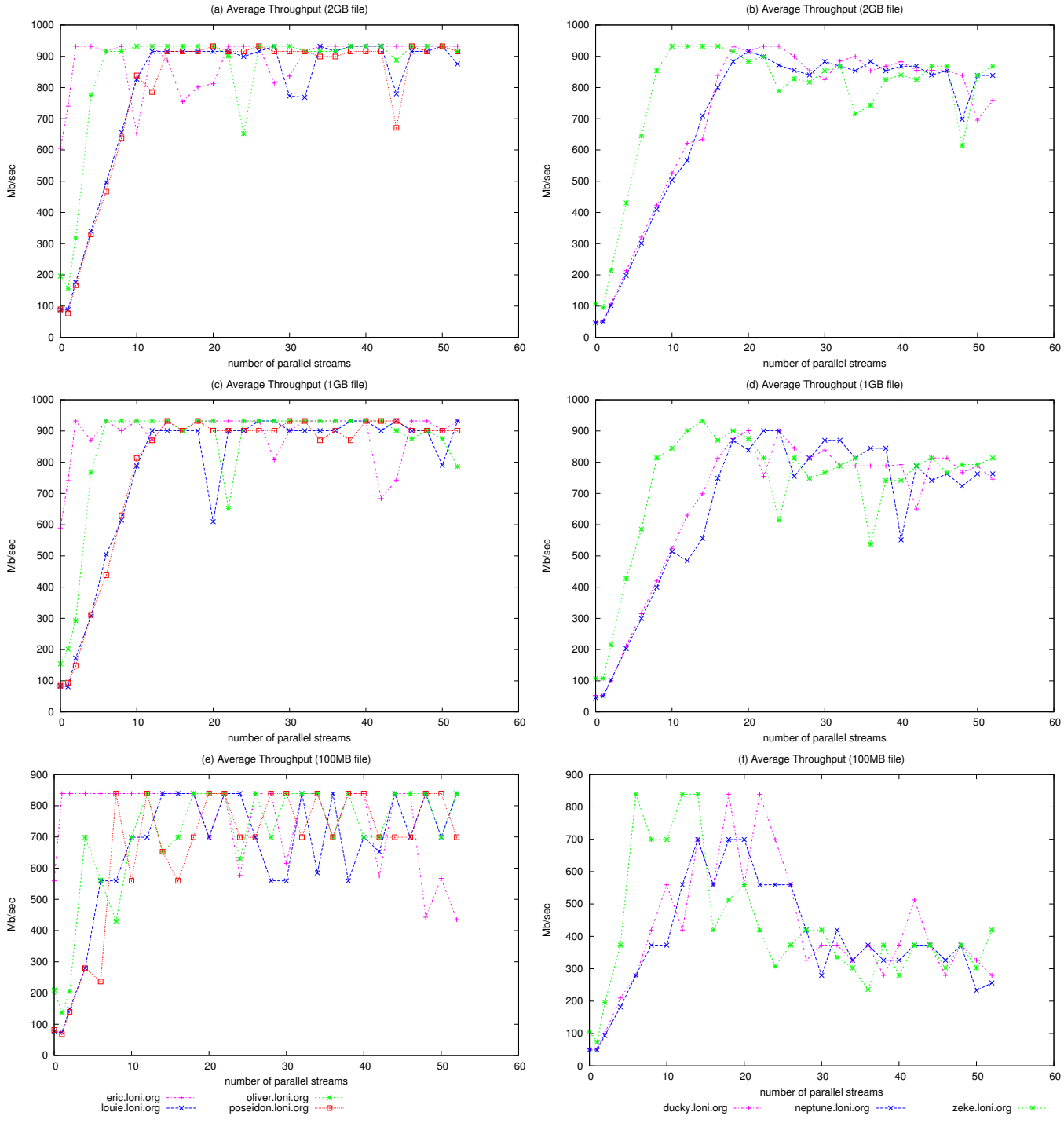


Figure 4.3: Average Throughput of file transfers to queenbee.loni.org using Parallel Streams. Get (a) 2GB file from Linux machines, (b) from IBM machines, (c) 1GB file from Linux machines, (d) from IBM machines, (e) 100MB file from Linux machines, (f) from IBM machines

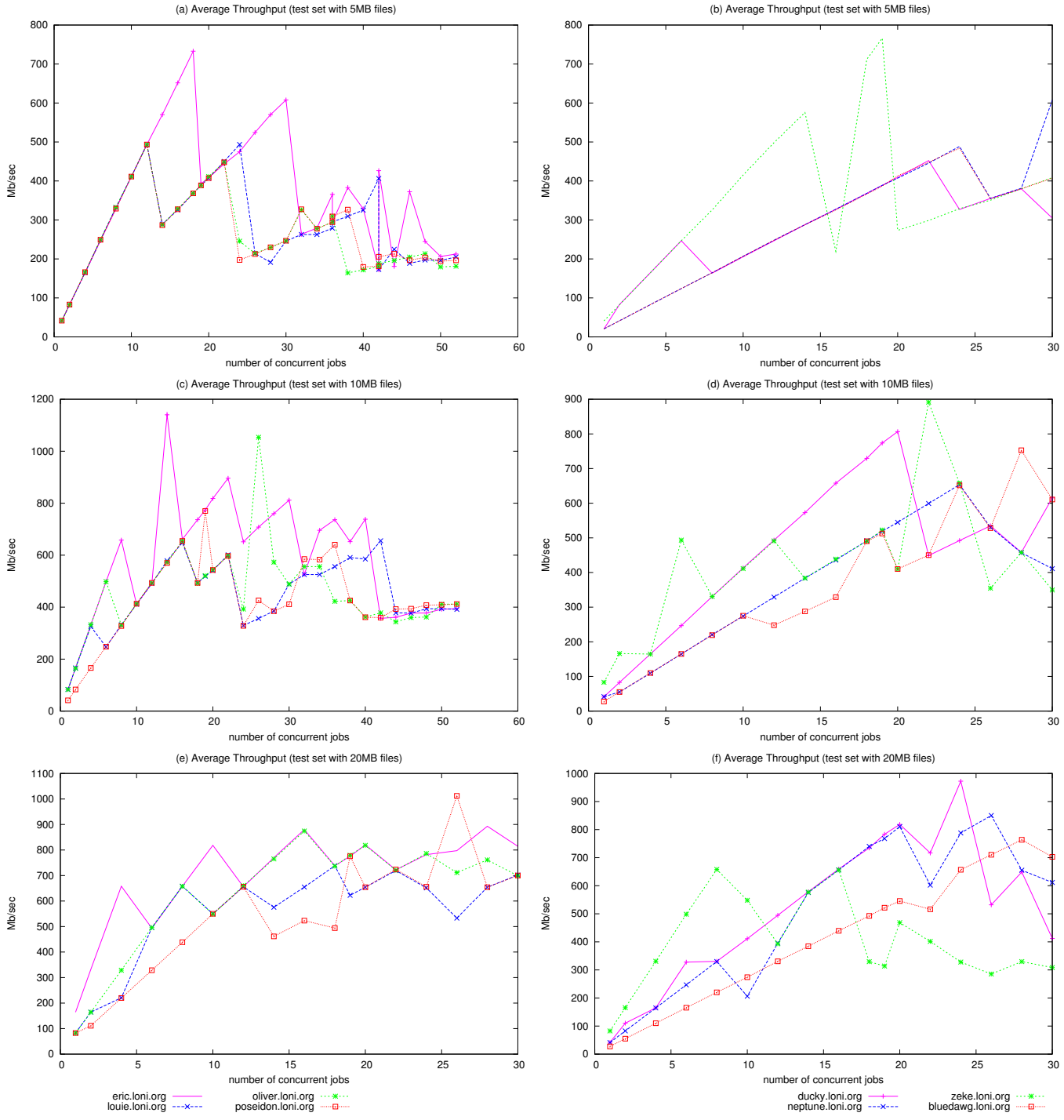
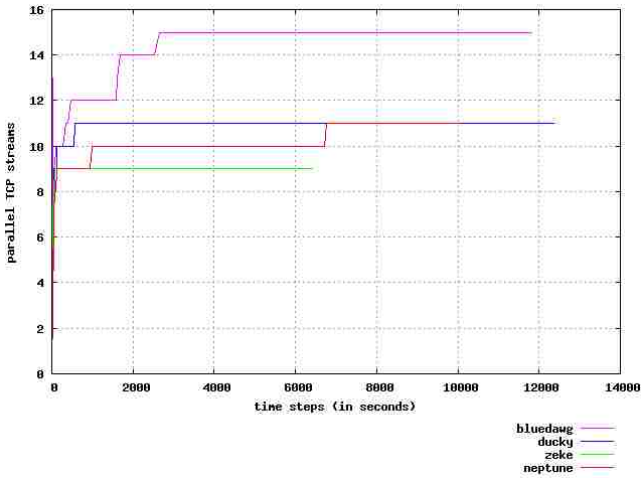
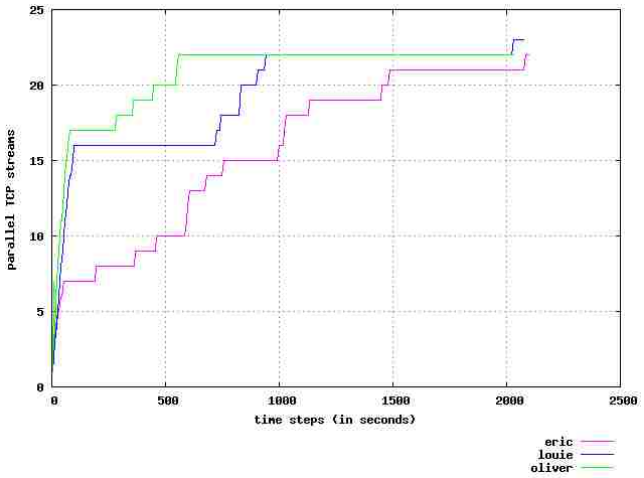


Figure 4.4: Average Throughput of Concurrent transfer jobs to queenbee.loni.org (getting multiple files concurrently). (a) with 5MB files from Linux machines, (b) from IBM machines, (c) with 10MB files from Linux machines, (d) from IBM machines, (e) with 20MB files from Linux machines, (f) from IBM machines

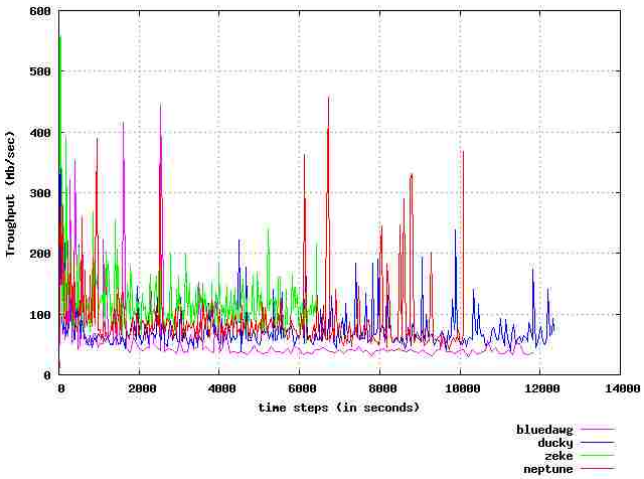


(a) get file from IBM machines to queenbee.loni.org

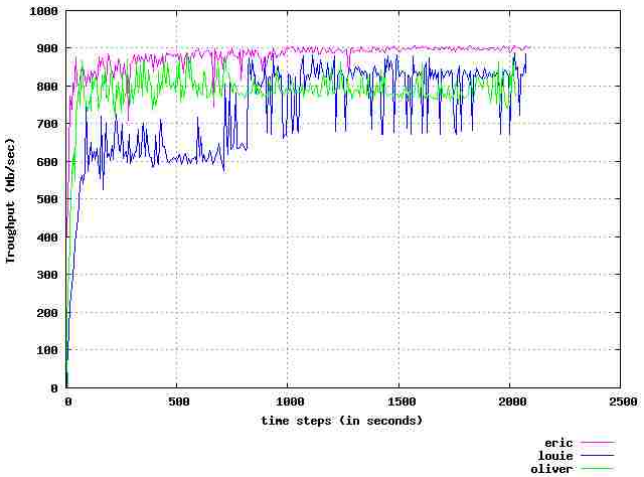


(b) get file from Linux machines to queenbee.loni.org

Figure 4.5: Dynamic Setting of Parallel Streams



(a) get file from IBM machines to queenbee.loni.org



(b) get file from Linux machines to queenbee.loni.org

Figure 4.6: Instant Throughput

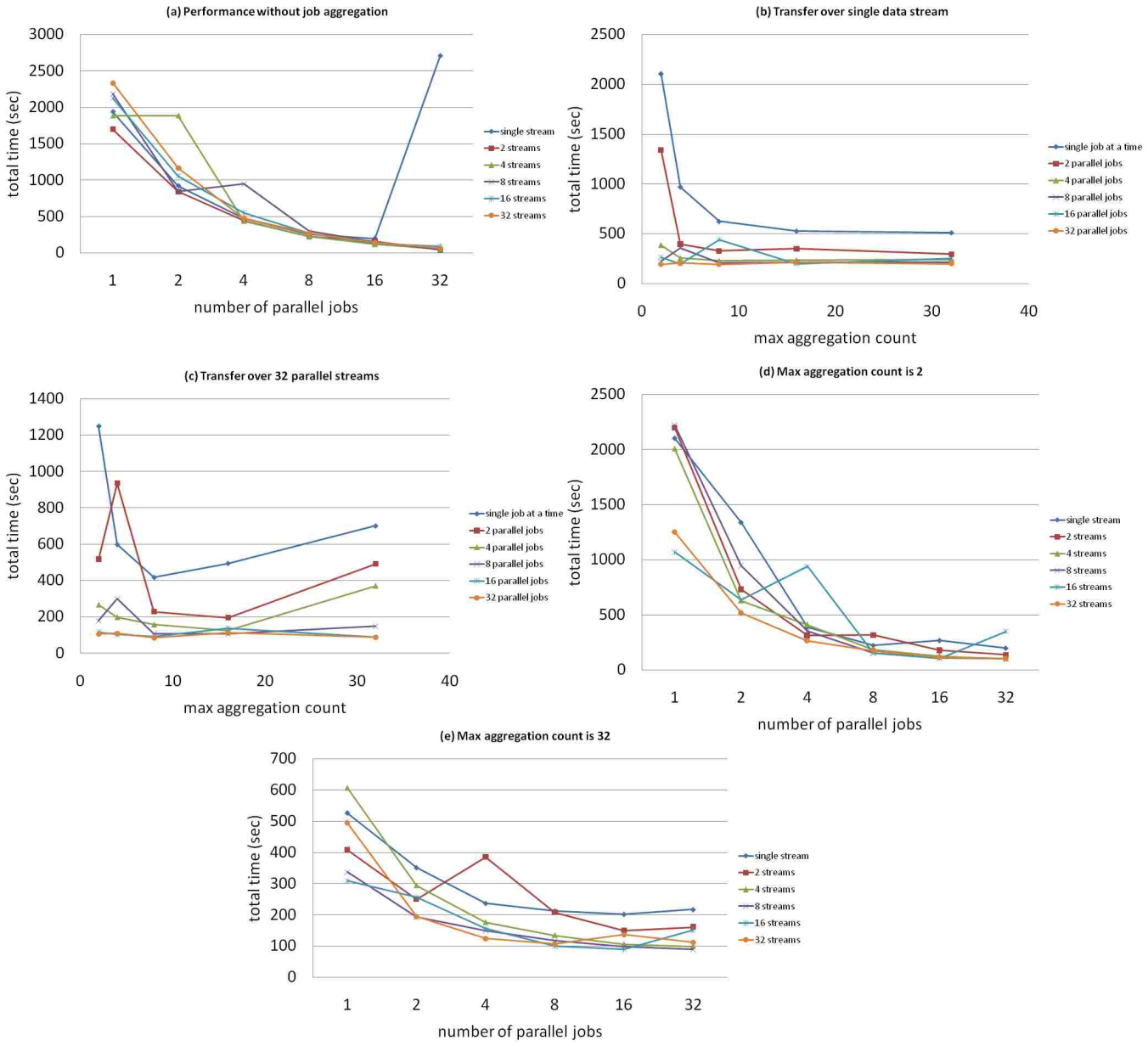


Figure 4.7: Performance measurement (a) without job aggregation; aggregation count vs number of parallel jobs: (b) transfer over single data stream, (c) transfer over 32 streams; number of parallel jobs vs number of multiple streams: (d) at most 2 jobs are aggregated, (e) at most 16 jobs are aggregated.

Chapter 5

Failure-aware Data Placement Scheduling

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 depends 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.

Distributed wide area networks differ from local area networks in terms of network topology, data transmission protocols, congestion management, latency, and bandwidth. High latency and limited bandwidth are the basic network characteristics affecting data transfer performance in distributed network environments. Moreover, communication protocols in distributed environments have some idiosyncrasies. Security, authentication and authorization are some of the other important issues in distributed data transfers. Since we deal with shared resources, even a simple file transfer over the Internet will be affected by many of the above factors, and if there is a high failure rate, we need to pay close attention. Hence, developing an efficient failure detection and recovery system for distributed networks is very crucial.

There has been many efforts to implement file transfer protocols over distributed environments conforming to the security framework of the overall system. These solutions should ideally exploit communication channel to tune-up network and to satisfy high throughput and minimum transfer time [112, 30, 62]. Parallel data transfers, concurrent connections, and tuning network protocols such as setting TCP buffer are some of the techniques applied. On the other hand, detecting an erroneous situation as early as possible before initiating the transfer, and reporting the reason of a failed transfer with useful information for recovery, should also be studied in order to supply better quality of service in distributed data transfers.

Large-scale scientific and commercial applications consist of many relevant tasks to be executed in geographically separated systems; such that, each stage retrieve information generated in previous stages

and output information to be used in following stages. In complex workflows where execution processes have data dependencies between each other, scheduling and ordering of data placement tasks not only enhances the overall performance but also prevents failures and inefficient resource sharing [81, 79]. Users usually do not have access to remote distributed resources and may not track the reason of a data transfer failure. If underlying protocol is unable to return useful information about the cause of a failed transfer, it is also inapplicable and not very useful for high level planners to develop fault tolerant structures.

Early error detection enables high level planners and workflow managers to have knowledge about a possible failure and a malfunctioning service in the environment. Instead of starting the data transfer job and waiting for failure to happen, those high level planners can simply search for another system or an alternative service to transfer data. Besides, classification and reporting of erroneous cases will help us to make better decisions.

The problem that we should consider 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 reporting. In error detection, we focus on making data placement scheduler aware whether destination host/service is available, and also making the scheduler able to select suitable data placement transfer services. We also explore techniques to trace an operation when transfer is in progress in order to detect failures and performance problems. 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.

In this chapter, we study possible methodologies to detect errors during data transfers and methods to efficiently report errors back to data placement schedulers. The outline of this chapter is as follows.

In Section 5.1, we explain possible methodologies for using network exploration techniques in early error detection. In Section 5.2, we propose a structural failure detection mechanism and failure-aware process cycle for data transfers. In Section 5.3, we analyze methods to keep track of operations while transfer is in progress, and we venture into how to use those methods in Stork data placement scheduler. In Section 5.4, we give details about our experiments and evaluation of proposed methods. And then, in Section 5.5, we put our conclusion, and open research problems in the area.

5.1 Early Error Detection with Network Exploration

Before initiating a data transfer operation in which source host will connect to 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 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 following five layers have been proposed in [74] as basic structure to make data intensive applications fault tolerant: (1) DNS resolve, (2) Host Alive, (3) Port Open, (4) Service Available, (5) Test service (transfer test data before starting the actual data placement). Normally, only root privileged users have access to ICMP layer which is used to detect whether remote host is up or down (ping utility). On the other hand, network scanners like Nmap [12, 13], are able to manage by directly accessing the Ethernet hardware using specialized libraries and not using the underlying network layer. We recognize the need to implement a service detection mechanism in which the following steps will be focused on: (1) DNS resolve, (2) Port Open, (3) Service Available (a simple test to examine functionality of the data transfer service).

We have experimented network exploration and service detection techniques and used Nmap features inside data placement operations to resolve host, scan predefined ports, and determine available services. Network exploration definitely puts extra overhead, but according to our experiments, shown in Section 5.4, it provides much quicker detection and recovery if compared with a failure reported by a transfer module without the ability of early error detection. One other possible drawbacks is that accessing network for detection may bother system administrators. However, we use limited set of exploration techniques to resolve host address and to explore given hosts also return available data transfer services. Therefore, our integration only includes special functionalities such that Stork scheduler transfer modules return with relevant error messages if availability of host and service is not justified.

Our proposed error reporting framework and a failure aware data transfer life cycle also benefits from network exploration tools. In Appendix B, we have explained network exploration in details and give infor-

mation about Nmap [12, 13], a network scanner tool, and then, we discuss our experiments with network scanner implementations and clarify applicability of those mechanisms in data placement schedulers.

5.2 Failure Detection and Error Classification

Every data transfer protocol comes with different methodology for initiating and processing the data transfer and also specific functionality in terms of authentication mechanism, protocol parameter control, and data channel usage. We are limited by the capability of the underlying data transfer protocol in order to get any information about a failure.

Although, data transfer protocols such as GridFTP notify if an error occurred and ensure the successful transmission of data, there is no generic error code to classify failure reason in every protocol. Besides, majority of those implementations are contented with returning error messages which are not specific and not explaining the situation in the environment causing this error.

As an example, a data transfer tool may return an error reporting that communication is aborted after a portion of a data file has been transmitted over the network. In such a case, there may be many reasons resulting in this failure; the remote host server may be down, or file transfer service is not functioning in the host, or file transfer service is not supporting some of the features requested, there may be a malfunctionality in the service protocol, or user credentials are not satisfied, or any other problem occurred in the source server.

Besides stating the problem with proper reasons, a higher level planner or a data transfer scheduler need information about the cause of a failure to perform the next action accordingly. If service or host server is not available temporarily, data transmission can be repeated afterwards; If there is no enough space in the remote server, another resource can be searched; if protocol is not supporting some features set to enhance performance, suitable parameters can be applied; another protocol or another service using the same protocol can be selected.

5.2.1 Framework for Error Detection and Classification

Our error reporting framework consists of generic operation types to capture information about progressing stages of every operation supported by different protocols. We classify operation types into 7 categories, as

can be seen in Figure 5.1. In the initialization phase, all parameters are set and connection is established to control the protocol. Information about supported features of the target data transfer service is gathered in feature select phase. In the configure phase, all parameters are set for tuning up the protocol or extending some supported features. Next, we check existence and status of files or directories in the remote or local data resources. Later, we perform the actual data transmit operation over the communication channel. After transfer operation has completed, some simple tests, like looking at checksum and comparing size both in source and destination, are performed to examine the successful transmission of data. Finally, there is finalizing operation to successfully close connections, and deactivate specific modules, and clear unused protocol handles.

Figure 5.1: Classification of Data Transfer Operations

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 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.

The specific error messages returned by data transfer protocols may not be useful to classify and report error cases. In our structural model, we define each operation in a different stage and with specific error codes returned to the scheduler. First, status information of a file will be examined, and then in the later stages, checksum or size of a file will be requested. *Error_Checksum_Mismatch* and *Er-*

ror_Filesize_Mismatch represent these type of errors in which we fail to verify the accuracy of the transferred data. In such a case, the scheduler may decide to retry transfering the file to ensure the accuracy of the transferred data. We initialize required transfer handles, activate modules specific to the protocol and prepare the system to connect and transmit information. A failure in the first stages shows that either the protocol is not supported or a proper connection can not be established. We represent this type of errors with the *Error_Protocol_Initialization* error code. If failure happens during the parameter configuration phase, different set of tune-up options can be used to accomplish a successful transfer. We use a specific error code, *Error_Unsupported_Features* to pass this information to the scheduler, so the transfer operation can be initialized with some other parameters if possible. User specific errors such as invalid file names, permission and authorization problems, or an attempt to get non-existing resource can be detected with this error code. Those type of error are not recoverable, so the scheduler will not retry this transfer operation and the operation will fail with error code *Error_User_Specific*. Figure 5.2 shows the interaction of operations in the error reporting framework.

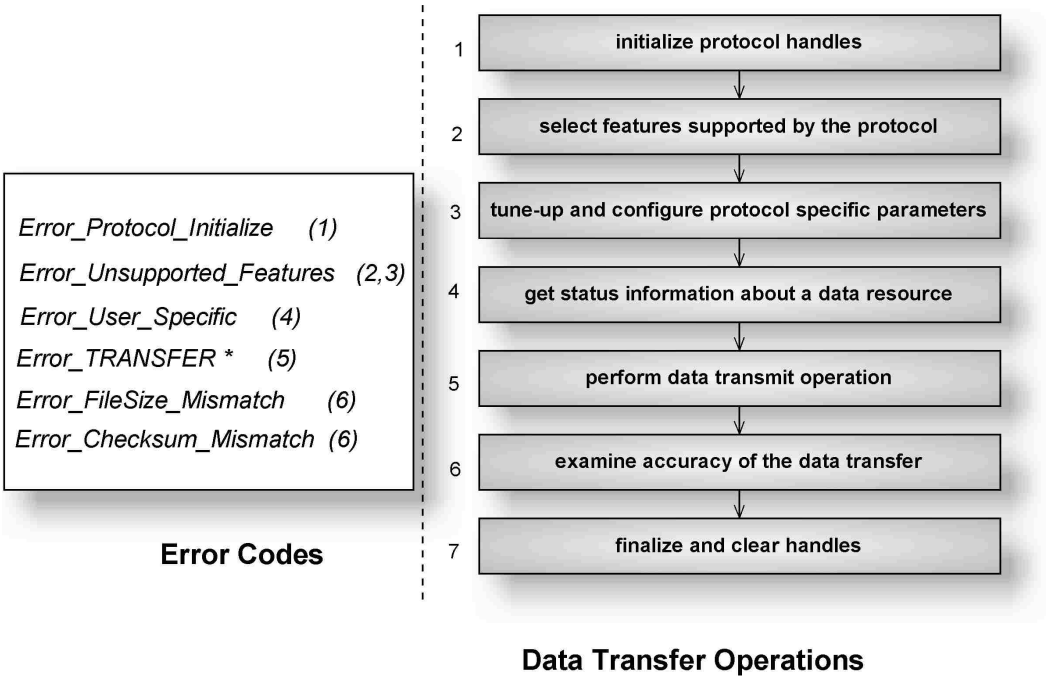


Figure 5.2: Interaction of Data Transfer Operation in the Error Reporting Framework

As an example, a directory transfer operation can fail if the file transfer protocol is not supporting directory listing. In such a case, transfer will fail with error code *Error_Unsupported_Features* 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 operation stage. In order to capture errors caused by network failures or mal-functionality in the protocol, we return the *Error_TRANSFER* error code. 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. An error condition may have different meanings whether it is before or after processing state in the operation. The network problem can be temporary, so the scheduler is supposed to retry the transfer operation after ensuring that erreneous condition has been recovered. An error condition may have different meanings whether it is before or after processing state in the operation. We give more details about this issue in the following section.

Categorizing possible operations in data transfers also provides more legible reporting in terms of users such that we do not need to deal with protocol specific error messages generated by different tools. On the other hand, data transfer tools and programming APIs are capable of reporting errors. Moreover, programming APIs and protocol specific tools are the best possible sources to get specific error messages. Thus, the proposed framework and categorization is not an alternative to the error reporting capabilities of the protocol specific tools or APIs. Rather, it is on top of them and using error messages generated to keep the condition in each phase.

5.3 Structural Failure Detection and Error Reporting

We propose to examine availability of the remote server and functionality of file transfer service before initiating the transfer. As it has been discussed in previous section, we can easily test whether we can access the remote host over the network. By using the network exploration techniques, we can also detect available services running on remote site. The *Error_Host_Down* error code is returned when the remote host is unaccessible over the network. Later, we check whether we can access the remote port that data transfer service is running on; the *Error_Port_Closed* is returned if the remote port is closed. One further step is to examine the functionality of the file transfer protocol such that we ensure it is responding as expected before starting the actual data transfer job. This can be accomplished by some simple file transfers

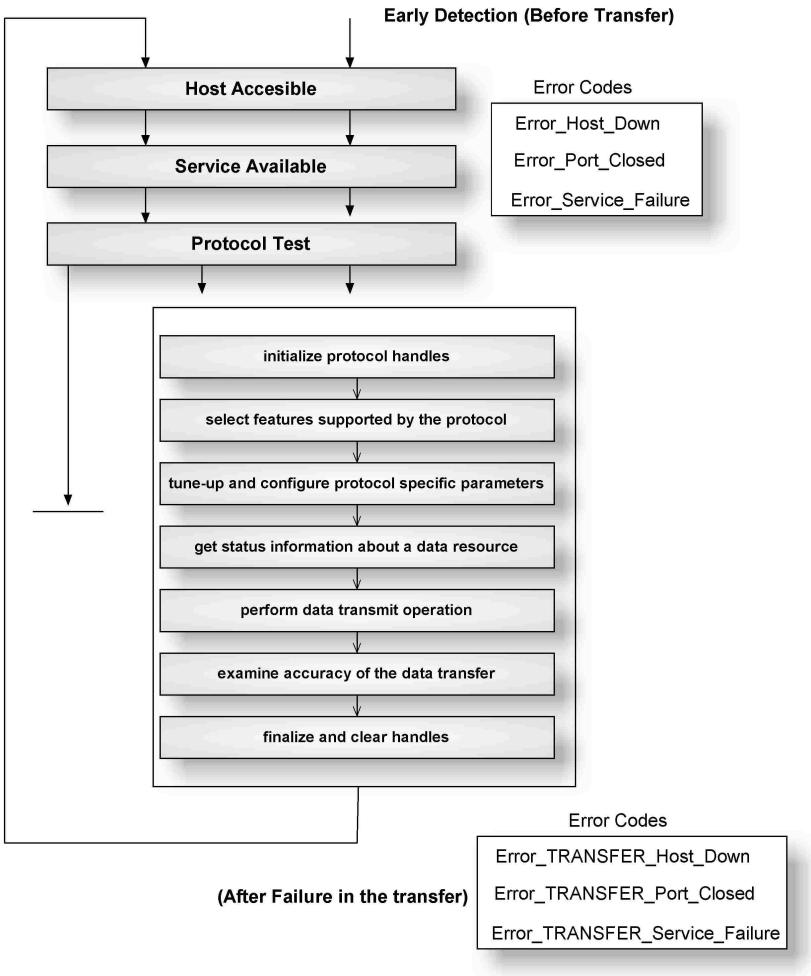


Figure 5.3: Error Reporting Framework for Data Transfer Operations

or by executing basic functions in the client interface of the protocol. The *Error_Service_Failure* error code is returned if the data transfer protocol is not responding according to its specifications.

A data transfer operation which passed all initial tests and which has been started, can fail after transferring some amount of data. In order to better understand the reason behind failure, we go further and perform initial tests again after an error occurred. We do not rely only on the error messages generated by transfer tools of client interfaces. As an example, a failure in a data transfer operations can be due to a network problem, host machine failure, or interruption in the service running on remote site. We explore the remote machine after a failure is received with the *Error_TRANSFER* error code. Applying network and service tests, and also protocol examination will enable us to decide on whether we should retry and start again the data transfer operation. Therefore, we have seperate error codes for network explorations tests applied to

remote host after a failed transfer.

First, we check whether we can access the remote machine on wide-area network. Determining the availability of the remote site does not bring serious overhead. In order to perform a remote file transfer, we need to activate the data transfer module, initiate the client interfaces, and connect to the service. Examining the network and then initiating the connection is much more efficient, if operation will fail due to a network problem.

After testing the connection availability, we perform service detection techniques such that we ensure the requested service is running in the remote site. This step has twofolds; we simply detect a possible failure due to unavailable service in the target host, and we can use the information of other available services to use an alternative protocol in data transfer.

The next step is checking whether remote service is functioning properly. This step has a crucial role in early error detection such that misconfiguration or any other problems in data transfer service can be detected here. Network failures other than server-related problems can be detected in previous stages, but if remote service is malfunctioning we do not need to wait to recover or to retry the operation.

We have prepared a testbed to detect possible erroneous cases in distributed data transfers and used GridFTP as our file transfer protocol. GridFTP client API [7, 32] provides an asynchronous mechanism such that operations are started and callback functions are called by the interface asynchronously. In order to prepare an error case which can not be reported and detected by current tools like globus-url-copy [6], we temporarily modified the configuration of the test server and change the hostname of the machine. GridFTP server program continued on execution properly and responded as expected to client calls provided by client interface. However, it has never responded with a callback to the client due to the misconfiguration in the server. GridFTP uses two communication channels; one for control operations and the other for data communication. One possible reason behind this situation is that data channel connection could not be established.

Globus-url-copy and client interface of GridFTP are not capable of detecting such an erroneous case. In our experiments, they both stacked and continued waiting for a callback call which will never be received. In our proposed structural framework, we test the connectivity and communication network beforehand. We are able to determine an approximate value about the round trip time and we can set a timeout for data

transfer operations. In the protocol test stage, if we cannot get response within this timeout limit, we can mark this service as faulty on the remote host.

As it has been described in section 5.2, it is also important to understand in which stage an error has occurred. If file transfer operation is interrupted due to host or service inaccessibility, the data scheduler or higher level planners can issue this operation later when error case has recovered. However, some other actions should be taken if there is a failure due protocol mis-functionality or permission error on the remote server. During our experiments with GridFTP, the most common error message returned by the client interface was closed connection by remote host. This error appeared not only for network problems but also certificate and permission issues in the remote system such that GridFTP server could not establish the data connection. In such a case, it is very much useful to perform initial network tests and understand the reason of the failure.

Therefore, we propose a structural error reporting framework in which data transfer operation is surrounded by particular tests scenarios. As it can be seen in Figure 5.3, we apply network and protocol tests even after a failed data transfer operation in order to classify erroneous cases. The purpose of those network and protocol examinations is not only detecting errors as early as possible but also reporting errors with as much detail as possible.

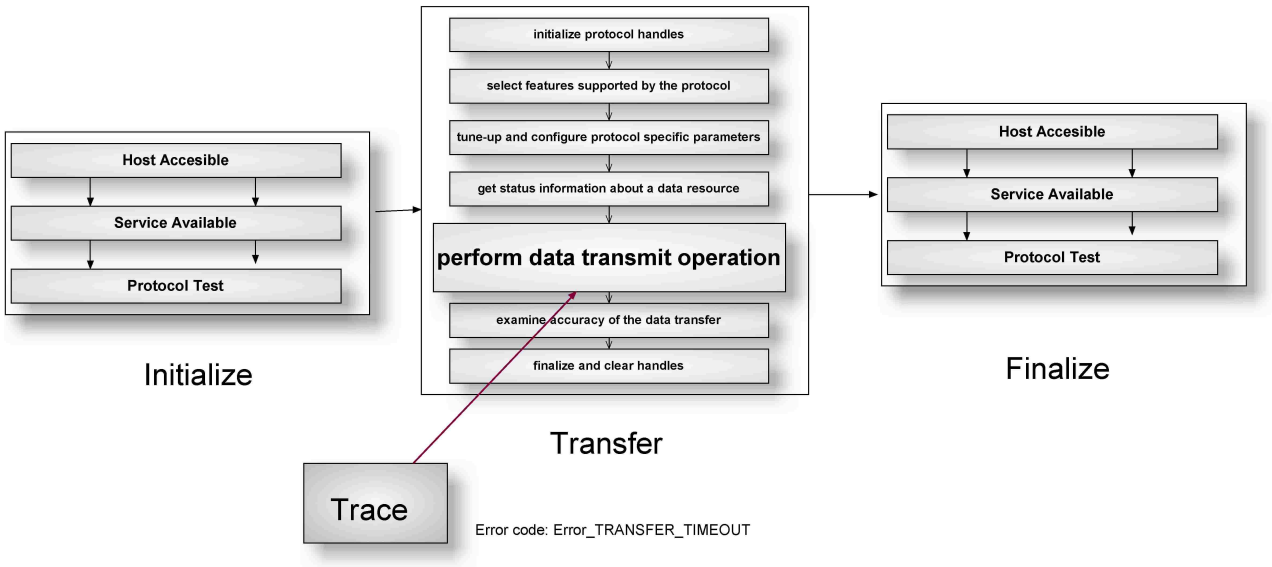


Figure 5.4: Data Transfer Life Cycle

5.3.1 Tracing Data Transfer Operations

Exploring performance and failure issues while transfer is in progress is quite difficult. We have discussed problems that can occur in the network connection and we studied possible solutions methodologies. As can be seen in Figure 5.4, we also perform tests for network and service availability at the end of a failed transfer in order to detect and mark a network problem. However, it is hard to trace if we have a problem while transfer is in progress. Therefore, we searched for possible techniques to trace a data transfer operation in order to detect failures and problems as soon as possible and report them to the data placement scheduler.

The GridFTP server may not return back any data in the data channel due to some misconfiguration or some other server side problem. If the callback function defined in GridFTP protocol is not executed, the client will hang since it has been waiting for an answer from the server. During the data transmit phase, which is defined in the structural error detection framework, we can benefit from tracing the running executable if we can capture useful information such as the amount of data transferred and the number of connections opened.

We have searched dynamic system tracing facilities and studied Dtrace [44, 45]. Dtrace brings the ability to dynamically instrument user-level and kernel-level application. It provides a C-like control language to define predicates and actions at a given point of instrumentation [45]. Dtrace architecture consists of basically providers, probes, actions and predicates. A probe is a programmable sensor such that it fires when the registered event happens [3]. If the predicate expression if validated, then the action is triggered. Providers offer the probes to the Dtrace framework such that they pass control to the Dtrace when a probe is registered [45, 3]. There are providers for monitoring scheduling service, system calls, I/O devices, and I/O requests, IP, virtual memory, etc [45].

Dtrace and similar tools have great importance in terms of dynamically tracing a running system. Many example scripts for performance monitoring are provided [3]; moreover, there is an effort on developing Dtrace network providers for network observability [45].On the other hand, there are also alternative tools like systemTap [22] in which we are also able to track reads and writes on socket initiated by the process.

The asynchronous property of GridFTP protocol makes it hard to track an operation when transfer is in progress. We first initiate a transfer operation and register read/write functions with appropriate callback functions, and then, wait callback functions to be executed. We studied instrumentation techniques to trace

our client application in order to understand the reason behind the error case where client hangs and waits forever. Using dynamic system tracking tools, we can see that there is no data transmission in progress, and we can make decision about the error condition.

Moreover, there are many other data transfer tools for specific protocol and we may not want to implement them from scratch according to the error reporting framework. Therefore, tracing the data placement application using external tools will enable us to capture the system calls, track network connections, and get information about the amount of data sent and received, and will help us to solve many other system related issues.

5.3.2 Integration of Failure-aware Scheduling

Scientific applications have become more data intensive like business applications; moreover, data management happens to be more demanding than computational requirements in terms of needed resources [64, 52]. Importance of data scheduling is emphasized by [81], and data has been stated as a first class citizen in the overall structure. Data placement is a coordinated movement of any information between related steps in the general execution workflow. Data placement is not limited to only information transfer over a network; however, one common use-case is to transfer data between servers in different domains using wide-area network. Data placement scheduler should be able to make rapid choices and dynamically adapt itself to changing situations.

We propose a virtual connection layer inside the data placement scheduler to keep track of network statistics. The data scheduler opens a virtual channel for every source-destination pair encountered so far, and updates network information such as host accessibility, available data transfer services. Whether an error condition has encountered, the status of the virtual channel is updated. Therefore, the scheduler can use this information and delay data transfer jobs which are requesting the virtual channel that has been marked as faulty.

The scheduler has a modular architecture such that data placement jobs are executed by external transfer modules [78]. Before initiating a transfer operation, we register for a virtual channel and update information about the connectivity and availability according to initial tests. Later, we also keep the statistics about failed jobs in the virtual channel. The concept of virtual communication channel defined as an abstract

level inside the data scheduler, enables us to utilize previous status information of failed operations. The error reporting framework and tracing transfer operations are sub-processes besides the main central data placement manager serving for better scheduling decisions. Figure 5.5 represents the overall structure of a data placement scheduler.

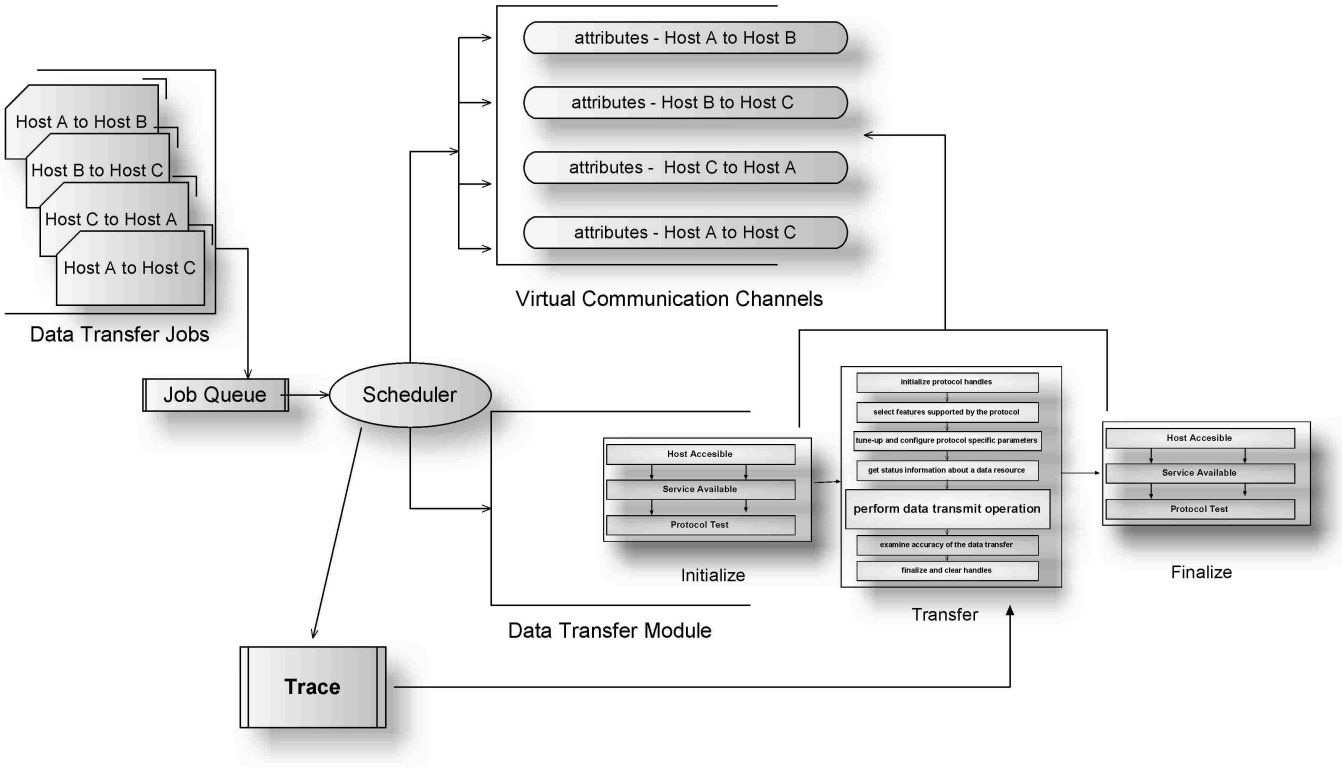


Figure 5.5: Integration of Structural Failure Detection and Error Reporting Framework into Data Placement Scheduler

5.4 Evaluation of Data Placement System Structure

We have prepared a testbed in which erroneous conditions are injected into data transfers to test our proposed structure. For our experiments, we used data files from the Scoop project [21, 97] for hurricane Gustav simulations.Our test environment includes LONI [9] machines and also one another server in the same network in which we have full administrative access to modify system specific attributes and generate errors for testing purpose. First, we examine the performance of early error detection system that is using network explorations techniques. We simple scheduled multiple data transfer operations with small (from 1MB to 100MB) and large files (from 1GB to 2GB) with and without early error detection module, and then, we

Table 5.1: Experiments with and without Network Exploration

Successful Transfers		Average Transfer time	Overhead of EarlyDetection
network rtt=0.548 ms	small files	4.505 (secs)	0.025 (secs)
	large files	32.245 (secs)	0.046 (secs)
network rtt=5.131 ms	small files	10.505 (secs)	0.36 (secs)
	large files	92.245 (secs)	0.46 (secs)

(a) Overhead of Early Error Detection for Successful Transfers

Error Recognition Time	without network exploration	with network exploration
firewall blocking	0.28 (secs)	0.001 (secs)
service unavailable	3.01 (secs)	0.001 (secs)

(b) Benefit of Network Detection Feature

take average values of total transfer times to examine the overhead of network exploration. As can be seen in Table 5.2(a), overhead is ignorable; besides, early error detection module provides faster recognition of the network error, shown in Table 5.3(b). We also would like to emphasize the benefit of accurate and more precise error classification system we have proposed to be utilized in decision making process of the data placement scheduler.

Next, we have experimented the impact of error detection and classification in data transfer scheduling. We used 250 data transfer jobs submitted to Stork scheduler and injected different types of errors into the system while the 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 5.6 show a heavily loaded queue in which all data transfer jobs are submitted initially. 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.

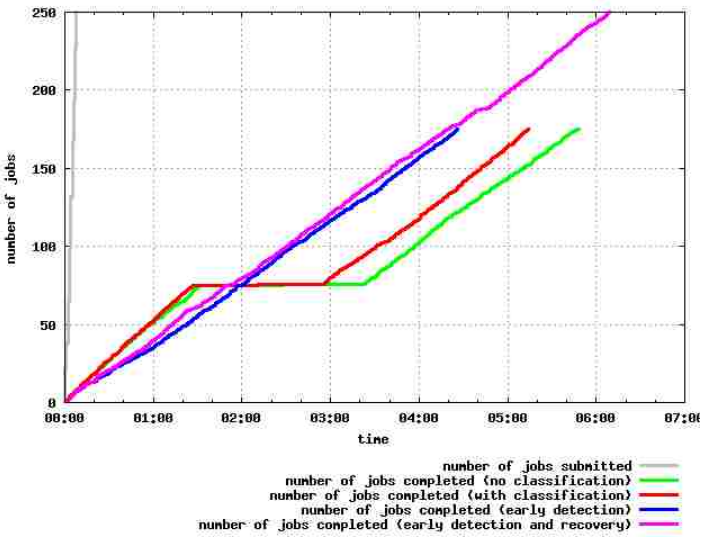


Figure 5.6: Evaluation of Error Detection and Classification

Stork, the data placement scheduler, checks network connection and availability of the data transfer protocol. We have implemented error detection and classification as new features inside Stork. Moreover, ee propose a generic framework and we have also tested our model with other data transfer protocols like iRods [17].

New data transfer modules are also able to verify the successful completion of the operation by controlling checksum and file size. Moreover, we have also implemented checkpointing for GridFTP operations such that Stork transfer module can recover from a failed operation by restarting from the last transmitted file. In case of a retry from a failure, 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 sample for Stork job submission is shown in the following:

```
[
  dest_url = "gsiftp://eric1.loni.org/scratch/user/";
  arguments = -p 4 dbg -vb";
  src_url = "file:///home/user/test/";
  dap_type = "transfer";
  verify_checksum = true;
  verify_filesize = true;
  set_permission = "755" ;
```

```
  recursive_copy = true;
  network_check = true;
  checkpoint_transfer = true;
  output = "user.out";
  err = "user.err";
  log =  "userjob.log";
]
```

Performance analysis shows that proposed framework does not put extra stress on transfer operations. The overhead incurred by error detection approach is negligible. Besides, network exploration methods provide much quicker detection and recovery if compared with a failure reported by a transfer module without the ability of early error detection.

5.5 Discussion on Failure Recognition

One interesting study investigating reasons behind failures in large scale production Grids uses data mining techniques to explore the relationship between failures and environmental properties [48]. Troubleshooting via data mining has been applied to diagnose reason behind a failure in real workloads, like many jobs running in a campus Grid. Failures have been classified according to execution environments, job and machine properties; such that, predefined decision points lead to correlations that are indicating main reasons of erroneous cases [48]. On the other hand, the proposed system does not aim immediate error detection. In our study, we focus on detecting erroneous cases on the fly and make scheduling decisions according to failure classification.

The importance of error propagation and categorization of errors in Grid computing has been mentioned clearly in [102]. This study builds a theory for error propagation which provides more robust distributed environment by considering the scope of errors. The new structure has three types; implicit, explicit and escaping errors [102]. An implicit error represents an invalid functionality. Explicit errors are due to an inability to accomplish the requested operation [102]. Explicit and escaping errors are connected to our focus in error classification; however, this error scope has been basically designed for Java Universe in Condor. We essentially concentrate on errors in data transfers for large scale applications.

A fault tolerant middleware has been described in [74] for data intensive distributed applications by examining the environmental conditions and classifying failures such that a suitable strategy can be applied to handle operations in a transient way. It uses log information from the job scheduler and the data place-

ment scheduler and takes the next action according to user policies [74]. We extend the error detection by including network exploration techniques and also proposed a better and more detailed classification methodology. In contrast, we explicitly focus on data transfers and our system structure handles operation in the perspective of data transfer scheduling without interfering the other components in the system.

Error detection and error classification have not been studied in detail for distributed data transfers. On the other hand, failure detection and error reporting are not only important issues in fault tolerant architectures, but also crucial in designing and planning the overall system architecture. We explore several mechanisms for early error detection and we define a structural framework for error classification. Moreover, we describe a data transfer process cycle in which main focus is to determine the erroneous situation in distributed data transfers. We have tested some of the features and implemented transfer modules to be used inside Stork, the data placement scheduler. In the near future, we are also planning to reshape Stork scheduling architecture according to the information gathered from detection steps and information obtained from error reporting framework. The data transfer life cycle explained in this study is intended to be the failure aware process cycle in Stork data placement scheduler. We have tested our system with 105,000 data transfer jobs for the movement of the Hurricane Gustav dataset from the Scoop project. From first hand experience, we believe that implementation of error detection and classification is unavoidable in large sets of data transfers. We underline once more the importance of developing failure aware data placement scheduling methodologies.

Chapter 6
Conclusion and Future Work

Scientific applications have become more data intensive; moreover, data management happens to be more demanding than computational requirements in terms of needed resources. 'Data-aware computing' is a new design paradigm integrating every element in the system hierarchy based on data access requirements of large-scale applications.

There has been several recent studies investigating new approaches for data management and data transfer in distributed systems. One approach is scheduling the distributed jobs close to the data in order to reduce the data transfer and I/O overhead [91, 93]. Another approach is to handle data placement jobs as a major element in the overall workflow affecting scheduling decision and the execution of compute jobs [79].

Prior knowledge about the environment, and awareness of the actual reason behind a failure, would enable the data placement scheduler to make better scheduling decisions. We have studied network exploration techniques in order to classify and detect network error as early as possible. Stork [79, 77], our data placement scheduler, checks the network connection and the accessibility of the data transfer protocol beforehand, with the help of a new network exploration module. This feature also enables us to select amongst the available data transfer services provided by a storage site.

Our experiments show that current data transfer protocols are not always able to generate adequate log information. Therefore, we focus on tracing the transfer job and preparing the infrastructure to explore dynamic instrumentation while transfer is in progress.

Latency between interconnects is one of the bottlenecks in data access - not only in distributed systems but also in microprocessor architecture. It is one of the reasons for an important limitation, memory wall, in microprocessor [58]. There are many studies that optimize network protocols to enhance wide area transfers by maximizing bandwidth utilization [110]. In addition to tuning TCP network parameters, there are also dynamic tuning tools for configuring buffer size and number of parallel streams on the fly [71, 60]. Even though it is considered as a TCP-unfriendly activity [75], tuning has its benefits. Our experiments emphasize the importance of tuning data transfers in wide-area networks.

Methodologies applied in BitTorrent-like peer-to-peer file sharing systems [46, 96] can also be integrated into data-aware computing for faster download of data chunks from multiple sources. We have been investigating the effects of multi-source transfers by implementing specific transfer modules in Stork. We have studied possible metrics affecting data placement in various scenarios [42]. We use historical data to tune up the network transfers and also to make better scheduling decisions based on those metrics [112].

On the other hand, trying to optimize these parameters all at once is a complex problem, especially in a case where there are multiple ongoing transfers. The data scheduler has the ability to collect information from multiple sites/links at the same time. It then uses the global knowledge to perform multi-parameter optimization to improve multiple transfers.

Since several data transfer tasks can share the common links and resources, we think that a virtual connection layer inside the data scheduler would help to keep track of network statistics for each data transfer. The virtual communication channel would enable the scheduler to apply network predictions to all transfers with a global knowledge of the system.

Reordering and aggregation of I/O requests have been studied in massively parallel architectures [68, 104]. Moreover, I/O forwarding and caching is crucial in supercomputers with many nodes in order to limit the number of I/O request sent to file servers.

We have successfully applied job aggregation in Stork 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. We measure the performance effect of the aggregation count (the maximum number of requests combined into a single transfer), along with the number of parallel streams and the number of simultaneous jobs running at the same time. The main performance gain comes from decreasing the amount of protocol usage and reducing the number of independent network connections. Thus, Stork makes better use of underlying infrastructure by coordinating and arranging data placement jobs.

Some recent studies are trying to reduce the amount of data to be 'actually' transferred between remote hosts. One method is exploiting the similarities between data objects using file handprints. The technique skips transfering chunks of a file that can be found in the local repository using constant number of lookups [88, 89]. Similarity detection is known in the literature [85]. Similar techniques are also implemented in archival storage [90, 50] such that the chunk level similarity is used to reduce copying duplicate blocks by computing checksum for each block.

One recent study [40] uses metadata information along with the semantic structure of the data to reduce the size of the actual data files, and to re-generate the data in the remote site instead of transferring the file from the data archive. Semantic compression has a broad perspective and it is successfully applied to compact database files [72]. Since data generated by scientific applications are usually structured, semantic analysis and semantic compression are also important research topics in Data Grids in terms of performance issues. We are still investigating possible approaches to exploit similarities and semantic analysis to be automatically applied in distributed data placement jobs inside the Stork scheduler.

Bibliography

[1] CMS: the US Compact Muon Solenoid Project. accessed in 2006 [Online] Available: http://uscms.fnal.gov/.

[2] CONDOR: High Throughput Computing. accessed in 2007 [Online] Available: http://www.cs.wisc.edu/condor/.

[3] Dtrace toolkit. accessed in 2007 [Online] Available: http://opensolaris.org/os/community/dtrace/dtracetoolkit/.

[4] FastTCP: an alternative congestion control algorithm in tcp. accessed in 2007 [Online] Available: http://netlab.caltech.edu/FAST/.

[5] FUSE: filesystem in userspace. accessed in 2008 [Online] Available: http://fuse.sourceforge.net.

[6] globus-url-copy: Multi-protocol data movement. www.globus.org.

[7] GT 4.0 GridFTP : Developer's Guide. accessed in 2007 [Online] Available: http://www.globus.org.

[8] Internet Secuirty: Port Scanning -Prabhaker Mate. accessed in 2007 [Online] Available: http://www.cs.wright.edu/ pmateti/Courses/499/Probing.

[9] Louisiana optical network initiative. accessed in 2008 [Online] Available: http://www.loni.org/.

[10] NetFlow: Cisco IOS NetFlow. accessed in 2007 [Online] Available: http://www.cisco.com.

[11] Network Probes Explained: Understanding Port Scans and Ping Sweeps - by Lawrence Teo. accessed in 2007 [Online] Available: http://www.linuxjournal.com/article/4234.

[12] Nmap. accessed in 2008 [Online] Available: http://insecure.org/nmap.

[13] Nmap Reference Guide (Man Page). accessed in 2008 [Online] Available: http://insecure.org/nmap/man.

[14] NWS: Network Weather Service. http://nws.cs.ucsb.edu/ewiki/.

[15] OptorSIM: A Grid simulator. http://www.gridpp.ac.uk/demos/optorsimapplet/.

[16] Parrot: Transparent Access to Grid Resources. accessed in 2008 [Online] Available: http://www.cse.nd.edu/ ccl/software/parrot.

[17] San Diego Supercomputer Center iRODS project. accessed in 2007 [Online] Available: https://www.irods.org/.

[18] Service and Application Version Detection. accessed in 2008 [Online] Available: http://insecure.org/nmap/vscan.

[19] STORK: A Scheduler for Data Placement Activities in the Grid. accessed in 2008 [Online] Available: http://storkproject.org/.

[20] Stream Control Transmission Protocol. accessed in 2007 [Online] Available: http://tools.ietf.org/html/rfc2960.

[21] SURA Coastal Ocean Observing and Prediction (SCOOP) Program. http://scoop.sura.org/.

[22] Systemtap. accessed in 2007 [Online] Available: http://sourceware.org/systemtap/.

[23] TCP/IP stack fingerprinting. From Wikipedia. accessed in 2008 [Online] Available: http://en.wikipedia.org.

[24] The Art of Port Scanning - by Fyodor. accessed in 2006 [Online] Available: http://insecure.org/nmap.

[25] Advanced Host Detection - Techniques To Validate Host-Connectivity. Synnergy Networks, 2001.

[26] CERN: the world's largest particle physics laboratory. accessed in 2006 [Online] Available: http://public.web.cern.ch, 2006.

[27] cFlowd: Traffic Flow Analysis Tool. accessed in 2007 [Online] Available: http://www.caida.org/tools/measurement/cflowd/, 2006.

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

[29] 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.

[30] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *IEEE Mass Storage Conference*, San Diego, CA, April 2001.

[31] 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.

[32] W. Allcock. Gridftp protocol specification, 2003. Global grid forum. GFD.20.

[33] 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.

[34] Gabrielle Allen, Kelly Davis, Thomas Dramlitsch, Tom Goodale, Ian Kelley, Gerd Lanfermann, Jason Novotny, Thomas Radke, Kashif Rasul, Michael Russell, Edward Seidel, and Oliver Wehrens. The gridlab grid application toolkit. In *HPDC*, page 411, 2002.

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

[36] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, and J. Semke. Ongoing TCP Research Related to Satellites., February 2000.

[37] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–10, 1990.

[38] Atlas. A Toroidal LHC ApparatuS Project (ATLAS). accessed in 2006 [Online] Available: http://atlas.web.cern.ch/.

[39] M. Baker. Ian foster on recent changes in the grid community. *Distributed Systems Online, IEEE*, 5:4/1–10, 2004.

[40] Pavan Balaji, Wu chun Feng, Jeremy Archuleta, Heshan Lin, Rajkumar Kettimuthu, Rajeev Thakur, and Xiaosong Ma. Semantics-based distributed i/o for mpiblast. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 293–294, New York, NY, USA, 2008. ACM.

[41] Mehmet Balman. Using Network Exploration and Service Detection Techniques in Stork-Data Placement Scheduler. http://csc.lsu.edu/ balman/technical.html.

[42] 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.

[43] 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.

[44] Bryan Cantrill. Hidden in plain sight. ACM Queue 4 (1): 26-36.

[45] Bryan Cantrill, Michael W. Shapiro, and Adam H. Leventhal. Dynamic Instrumentation of Production Systems. Proceedings of the 2004 USENIX Annual Technical Conference.

[46] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, and Atul Singh Antony Rowstron. Splitstream: High-bandwidth content distribution in a cooperative environment, 2003.

[47] Wu chun Feng, Mike Fisk, Mark K. Gardner, and Eric Weigle. Dynamic right-sizing: An automated, lightweight, and scalable technique for enhancing grid performance. In *PIHSN '02: Proceedings of the 7th IFIP/IEEE International Workshop on Protocols for High Speed Networks*, pages 69–83, London, UK, 2002. Springer-Verlag.

[48] David Cieslak, Nitesh Chawla, and Douglas Thain. Troubleshooting Thousands of Jobs on Production Grids Using Data Mining Techniques. *IEEE Grid Computing*, September 2008.

[49] Marco Conti, Emmanuele Monaldi, and Andrea Passarella. An adaptive data-transfer protocol for sensor networks with data mules. In *Proc. IEEE International Symposium on a World of Wireless, Mobile, and Multimedia Networks (WoWMoM 2007)*, 2007.

[50] Landon P. Cox, Christopher D. Murray, and Brian D. Noble. Pastiche: making backup cheap and easy. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 285–298, New York, NY, USA, 2002. ACM.

[51] 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.

[52] DOE. Center for Enabling Distributed Petascale Science. *A Department of Energy SciDAC Center for Enabling Technology*, 2006.

[53] T. Dunigan, M. Mathis, and B. Tierney. A TCP Tuning Daemon. In *Proceedings of the SC02: High Performance Networking and Computing Conference*, 2002.

[54] Tom Dunigan, Matt Mathis, and Brian Tierney. A tcp tuning daemon. In *in Proceedings of SuperComputing: High-Performance Networking and Computing*, 2002.

[55] Phillip Dykstra. High performance data transfer. supercomputing 2006 tutorials M07, November 2006.

[56] Lars Rene Eggert, John Heidemann, and Joe Touch. Effects of Ensemble-TCP, January 2000.

[57] S. Floyd. Congestion control principles. IETF RFC 2914. http://www.ietf.org/rfc/rfc2914.txt, 2000.

[58] M. J. Flynn. Basic issues in microprocessor architecture. *J. Syst. Archit.*, 45(12-13):939–948, 1999.

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

[60] Mark K. Gardner, Wu chun Feng, and Mike Fisk. Dynamic right-sizing in ftp (drsftp): Enhancing grid performance in user-space. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 42, Washington, DC, USA, 2002. IEEE Computer Society.

[61] Mark K. Gardner, Sunil Thulasidasan, and Wu chun Feng. User-space auto-tuning for tcp flow control in computational grids. *Computer Communications*, 27:1364–1374, 2004.

[62] GridFTP. Protocol extensions to ftp for the grid. accessed in 2006 [Online] Available: http://www.globus.org.

[63] John L. Hennessy and Norman P. Jouppi. Computer technology and architecture: An evolving interaction. *Computer*, 24(9):18–29, 1991.

[64] 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.

[65] Stephan Hirmer, Hartmut Kaiser, Andre Merzky, Andrei Hutanu, and Gabrielle Allen. Generic support for bulk operations in grid applications. In *MCG '06: Proceedings of the 4th international workshop on Middleware for grid computing*, page 9, New York, NY, USA, 2006. ACM Press.

[66] K. Holtman. CMS data grid system overview and requirements. *CMS Note 2001/037,CERN*, 99, July 2001.

[67] Randall Hyde. *The Art of Assembly Language Programming - CPU Architecture*. No Starch Press, 2003. webster.cs.ucr.edu/AoA/Windows/PDFs/CPUArchitecture.pdf.

[68] Kamil Iskra, John W. Romein, Kazutomo Yoshii, and Pete Beckman. Zoid: I/o-forwarding infrastructure for petascale architectures. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 153–162, New York, NY, USA, 2008. ACM.

[69] Takeshi Ito, Hiroyuki Ohsaki, and Makoto Imase. On parameter tuning of data transfer protocol gridftp in wide-area grid computing. In *Proceedings of Second International Workshop on Networks for Grid Applications, GridNets 2005*, pages 415–421, 2005.

[70] Takeshi Ito, Hiroyuki Ohsaki, and Makoto Imase. Automatic parameter configuration mechanism for data transfer protocol gridftp. In *SAINT '06: Proceedings of the International Symposium on Applications on Internet*, pages 32–38, Washington, DC, USA, 2006. IEEE Computer Society.

[71] Takeshi Ito, Hiroyuki Ohsaki, and Makoto Imase. Gridftp-apt: Automatic parallelism tuning mechanism for data transfer protocol gridftp. *ccgrid*, 00:454–461, 2006.

[72] H. V. Jagadish, Raymond T. Ng, Beng Chin Ooi, and Anthony K. H. Tung. Itcompress: An iterative semantic compression algorithm. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 646, Washington, DC, USA, 2004. IEEE Computer Society.

[73] 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.

[74] G. Kola, T. Kosar, and M. Livny. Phoenix: Making data-intensive grid applications fault-tolerant. In *5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[75] G. Kola, T. Kosar, and M. Livny. Profiling grid data transfer protocols and servers. In *Proceedings of 10th European Conference on Parallel Processing (Europar 2004)*, Pisa, Italy, August 2004.

[76] George Kola, Tevfik Kosar, and Miron Livny. Run-time adaptation of grid data placement jobs. In *In Proceedings of Int. Workshop on Adaptive Grid Middleware*, 2003.

[77] 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.

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

[79] Tevfik Kosar. A new paradigm in data intensive computing: Stork and the data-aware schedulers. In *Challenges of Large Applications in Distributed Environenments (CLADE 2006) Workshop*. HPDC 2006, June 2006.

[80] 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, New York, NY, USA, 2004. ACM Press.

[81] Tevfik Kosar and Miron Livny. Stork: Making data placement a first class citizen in the grid. In *In International Conference on Distributed Computing Systems*, March 2004.

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

[83] 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.

[84] 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.

[85] U. Manber. Finding similar files in a large file system. In *Proc. of the Winter 1994 USENIX Conference*, pages 1–10, San Francisco, CA, 1994.

[86] 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.

[87] 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.

[88] Himabindu Pucha, David G. Andersen, and Michael Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Cambridge, MA, April 2007.

[89] Himabindu Pucha, Michael Kaminsky, David G. Andersen, and Michael A. Kozuch. Adaptive file transfers for diverse environments. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2008.

[90] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *FAST '02: Proceedings of the Conference on File and Storage Technologies*, pages 89–101, Berkeley, CA, USA, 2002. USENIX Association.

[91] Ioan Raicu, Yong Zhao, Ian Foster, and Alex Szalay. Accelerating large scale scientific exploration through data diffusion, 2008.

[92] 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.

[93] 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.

[94] 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.

[95] Root. Object Oriented Data Analysis Framework, 2006.

[96] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie: A cooperative bulk data transfer protocol, 2004.

[97] B.H. Stamey, V.Wang, and M. Koterba. Predicting the next storm surge flood. *Sea Technology*, pages p10–15, August, 2007.

[98] Jon M. Stokes. *Inside the Machine - An Illustrated Introduction to Microprocessors and Computer Architecture*. No Starch Press, December 2003, 320 pp.

[99] Ibrahim H Suslu, Fatih Turkmen, Mehmet Balman, and Tevfik Kosar. Choosing between remote i/o versus staging in large scale distributed applications. *Parallel and Distributed Computing and Communication Systems*, 2008.

[100] 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.

[101] 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.

[102] Douglas Thain and Miron Livny. Error scope on a computational grid: Theory and practice. In *In Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing*, pages 199–208. IEEE Computer Society, 2002.

[103] 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.

[104] Rajeev Thakur and Ewing Lusk. Data sieving and collective i/o in romio. In *In Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pages 182–189. IEEE Computer Society Press, 1999.

[105] Brian L. Tierney, Jason Lee, Brian Crowley, Mason Holding, Jeremy Hylton, and Fred L. Drake, Jr. A network-aware distributed storage cache for data-intensive environments. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, pages 185–193, Redondo Beach, CA, 1999. IEEE Computer Society Press.

[106] 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.

[107] 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, New York, NY, USA, 2004. ACM Press.

[108] Kaladhar Voruganti, M. Tamer Özsu, and Ronald C. Unrau. An adaptive data-shipping architecture for client caching data management systems. *Distrib. Parallel Databases*, 15(2):137–177, 2004.

[109] Ofri Wechsler. Inside Intel Core Microarchitecture - Setting New standards for Energy Efficient Performance, 2006. http://download.intel.com/technology/architecture.

[110] E. Weigle and W. Feng. A comparison of tcp automatic tuning techniques for distributed computing, 2002.

[111] Eric Weigle and Wu-chun Feng. Dynamic Right-Sizing: A Simulation Study. In *10th International Conference on Computer Communication and Networking (ICCCN 2001)*, Scottsdale, Arizona, October 2001.

[112] 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.

Appendix A
Data Management in Computer Architecture

Early chip architectures were designed only for specific purposes. The von Neumann architecture brought the stored program model in which instructions are also stored as data. The basic von Neumann architecture, shown in Figure A.1, simply consists of a control unit designed to interact with other devices in the computer, an arithmetic logic unit designed for calculations, and a memory unit storing both data and instructions. Intel 8086 8-bit processor was one of the first processors designed based on stored programming model [98, 63]. It basically consists of an execution unit and an address translation unit to send and receive data from memory.

A.1 Microprocessor Evolution

In the early models, accessing data was one of the major problems as it is still a crucial issue today; such that, data retrieval from memory to execution unit is slow due to the speed of memory and also the latency in the interconnect between memory and CPU. Figure A.2 shows a very fundamental model for the processor organization.

The x86 line processors presented a new concept known as out order processing for this latency problem [67]. Instructions include load and store operations to or from memory to retrieve data which is associated with the execution of operands. An out of order processor has a re-order buffer to store instructions, so some of them can be executed before others in the given sequence; and, such a straightforward ordering can improve the memory latency. Memory access operations (load/store) are supposed to take more CPU

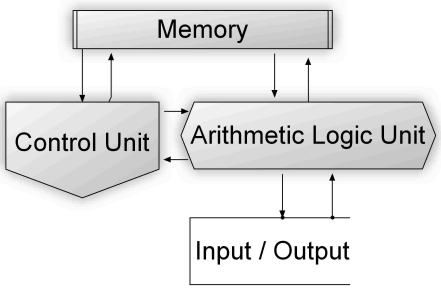


Figure A.1: von Neuman Architecture

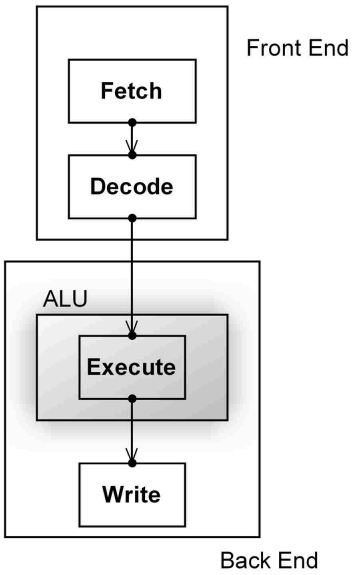


Figure A.2: Basic Instruction Flow

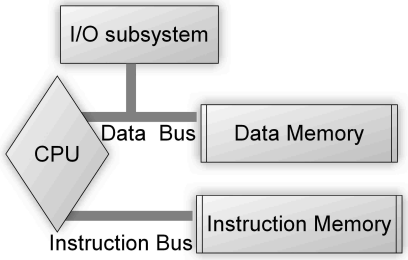


Figure A.3: Harvard Architecture

cycles than arithmetic operations (add/multiply), so instead of wasting CPU time some other operands are executed while waiting for the data to be retrieved.

The Harvard Architecture, shown in Figure A.3, is a model in which separate memory spaces are used for instructions and data, so there will be no contention for the communication bus [98]. Cache system is one of the main improvements reducing the waiting time for memory access. Most of the memory accesses are repetitive; therefore, instructions and then the necessary data for those instructions can be fetched to memory before they begin to execute. Possible instructions that might be executed in the next few cycles are stored into L1 or L2 cache waiting to be accessed by the CPU without any memory latency. Moreover, there has been a lot effort on improving caching algorithms and current caching methodologies are quite effective such that almost accurate branch predictions can be made by pre-fetchers.

Another enhancement in microprocessors is the SIMD instructions in which a special single intruction is

run over multiple data items. The SIMD was available with Intel MMX technology. Then, streaming SIMD extensions (SSE) were introduced. System performance become better with faster chips and we started to use multiple cores. Hyper Transport has been introduced to enable high bandwidth and high bandwidth communication data bus between multiple cores [109]. Another improvement was the Integrated Memory Controller in which the memory controller is directly integrated inside the architecture.

Microprocessors have evolved and changed quietly [63]. Besides, the complexity of processors with many internal layers has resulted in more efficient and fast computing even at lower clock speeds. For example, Intel multi-core architecture includes many layers such as instruction fetch, decode, instruction queues, dispatch unit and reorder buffer, and separate units for floating point, integer and load/store operations [67, 98]. New generation CPUs, though they are CISC architectures, also use the pipeline concept heavily to provide more number of instructions executed per cycle. Execution cores contains separate units to allow more than one instruction simultaneously; as an example, Intel core has fetch, decode, execute, and retire units, so up to four full instruction can be executed at the same time.

One important issue is that data load/store operations are dealt with different parts in the overall system. Efficient data access is maybe the most important concern effecting performance. Cache management algorithms and memory access approaches like Intel Smart Memory Access and utilization of L1 and L2 caches enhance the performance by reducing the latency. Intel's memory disambiguation architecture uses special algorithms to predict whether a load instruction depends on previous store operation, so it can be scheduled before preceding store instructions to obtain the highest level of parallelism [98, 109].

A.2 Limits in Microprocessor Architecture

Program behavior and memory access have increasingly become more important in microprocessor architecture design as die density and processor speed increase due to developments in silicon technology. Microprocessors evolved very rapidly and computing power improved more than estimated projections in past years [58], but microprocessor architecture is limited by physical device barriers and also practical bounds. Since physical limits have almost been reached, processor design should be implemented considering the current technology in hardware and the knowledge about user behaviour [58].

We are able to implement significantly more devices on a given circuit area such that improvement in

reducing the feature size enabled smaller device dimensions and the area a chip occupies decreases by the square of the scaling factor [58]. On the other hand, device delay is linearly related with the feature size [58]. As feature size shrinks, device size shrinks as the square of the feature size, and device speed improves linearly by feature size [58]. The problem is that interconnect delays between any two points does not scale linearly as it has been observed in device speed. The delay associated with local interconnects hinders the increase in microprocessor speed, and this leads to new designs in which more compact functional units with minimum interconnections are used and architectural structures are implemented focusing on minimizing the effect of local interconnect delay [58, 67, 63].

Due to lack of linear scaling in interconnect delay as feature size shrinks, clock speed has increased much slower than it happened in the past [58]. Another important limit in microprocessor architecture is memory access time. Availability of smaller devices enabled more memory per chip. Memory density and capacity increased excessively. However, larger memory means that interconnect lines should cover larger number of devices and memory access time is likely to be limited by local interconnects. Memory access time has not improved much as memory chip capacity increased due to poor scaling of interconnects[67, 58]. Therefore, hierarchical memory systems, cache and buffer subsystems have been developed to match the increase in processor cycle time. Since memory access time almost remained constant during the evolution of microprocessors, there have been several solutions in memory structure design. Instead of having one big memory array, it has been implemented in such a way that memory capacity has been divided into multiple arrays to reduce the effect of interconnect delays [58].

Memory access time is a crucial bottleneck since number of instructions executable in a memory access time increases as processor cycle time improved and enhanced instruction level parallelism has been used [63]. There is a maximum logical memory latency such that applications will see no performance beyond that memory wall [58]. Therefore, overall performance is limited with memory latency and bandwidth. One solution is large cache usage to minimize the hit rate and with large cache line sizes, so required memory bandwidth is increased.

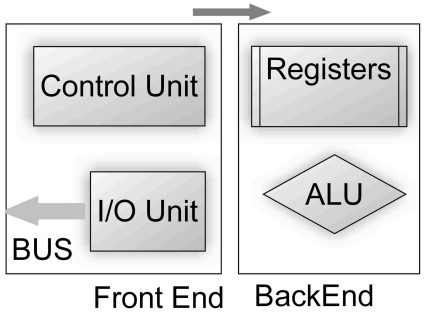


Figure A.4: Front End / Back End

A.3 Discussion on Microprocessor Design

One of the key innovations is the pipelined execution which characterizes the performance increase in the development of microprocessor design. Instruction execution has been divided into sub-steps to increase the operational speed of each step using specialized execution units, so it does speed up the program execution time. The lifecycle of an instruction consist several steps [98]. First, instructions are fetched and decoded in the instruction register. Then, they are sent to the ALU to perform the coded operation. Finally, result is written back to the register file. In terms of architectural division, an instruction starts in the fetch phase, moves to decode phase, and it is sent to the execution phase, then to the write phase.

Conceptually, a microprocessor is divided into two parts as front-end and back-end [98], shown in FigA.4. Front-end corresponds to control and I/O units which are communicating with the main memory. Instructions flow through back-end from front-end to enter write and execute phases in the pipeline [98]. The basic idea in a pipelined architecture is increasing the throughput of instruction processing by using small stages. Memory-access is the main cause of pipeline stalls [98]. While waiting data from main memory, processor can complete many instructions. Therefore, caching techniques have been used to meet the speed difference between main memory hardware. Another issue is the case where branch instruction are encountered such that pipeline should be refilled. In order to get the targeted performance, pipeline stages should always be filled up with instructions. Thus, new techniques like branch prediction, instruction and data caching are used to support pipeline architecture which has been designed to gain performance by exploiting the sequential flowing nature of instructions.

Separate integer execution units and separate memory-access units have been used to exploit the super-

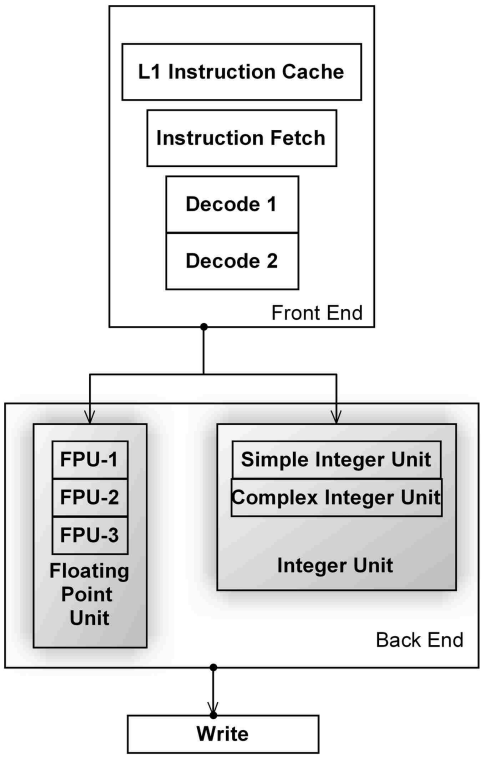


Figure A.5: The Pentium's pipeline [98]

scalar design and increase the performance. Figure A.5 gives the overall structure of Pentium pipeline [98]. Memory-access instructions like load, store and branch are handled with separate units inside the microprocessor. Load/store units are responsible for the transfer of data from and to memory and also responsible for address generation. Likewise, branch execution unit is used for conditional and unconditional branch instructions such that instructions are fetched beforehand.

There has been a speed gap between memory and processor such that many processor cycles are required to load data from main memory into registers and execution units. Besides, very fast memory is quite expensive. As a result, smaller amount of memory -cache memory - which is faster and expensive have been placed between processor and main memory to fill the speed gap. Usually, there are multiple cache layers; i.e. L1 cache is the smallest and fastest memory closest to the processor [98].

Additionally, execution stages are split into execution units and also into multiple pipelines [98]. Instructions are dispatched to multiple execution units by static scheduling in which hard-coded rules are used to check if they can be processed in parallel. Instructions from decode stage flow into a buffer where they wait to be scheduled for optimal execution into the execution units. Buffering instructions and reordering them

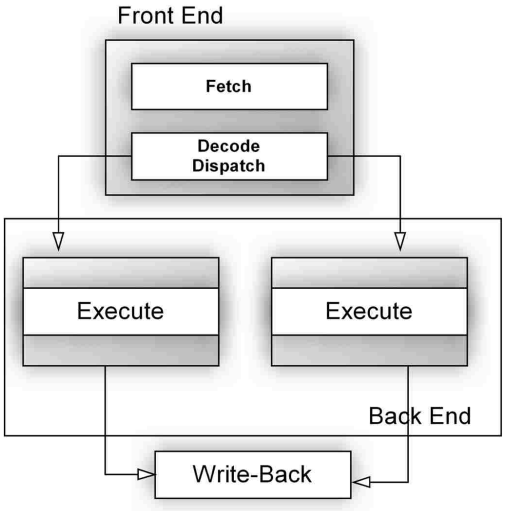


Figure A.6: Static Scheduling [98]

to be sent to the execution units out of order is called dynamic scheduling, and this has been accomplished by buffering before the execution phase [98]. We access register files in parallel in a pipeline architecture despite the fact that they are given in order in the program. Therefore, hidden register files are used such that results are committed in the final write phase. Figure A.6 and A.7 gives details about static scheduling and dynamic scheduling with issue buffers. Figure A.9 represents the basic architecture of Pentium 4 [109, 98].

The Pentium Pro architecture, shown in Figure A.8, has a special branch unit in the front end. Instructions are fetched and decoded, and then sent to the reorder buffer. Reservation station inside the back-end dispatch instructions to sent to separate execution units. By the release of Pentium Pro, a separate load store unit is used [109]. The fetch/decode bandwidth of front-end and execution bandwidth of back-end are separated with this buffering methodology. In P6 architecture, reorder buffer has 40 entries; and in each field, there is a tracking entry field to keep the original program order, and a data field to keep the result for register renaming [109, 98]. Moreover, reservation station can hold up to 20 instructions to search for the optimal scheduling decision among those instructuons. Intel started to use larger L1 cache in Pentium II [98]. Larger cache enhanced the performance by keeping the pipeline full. Figure A.9 represents the basic architecture of Pentium 4.

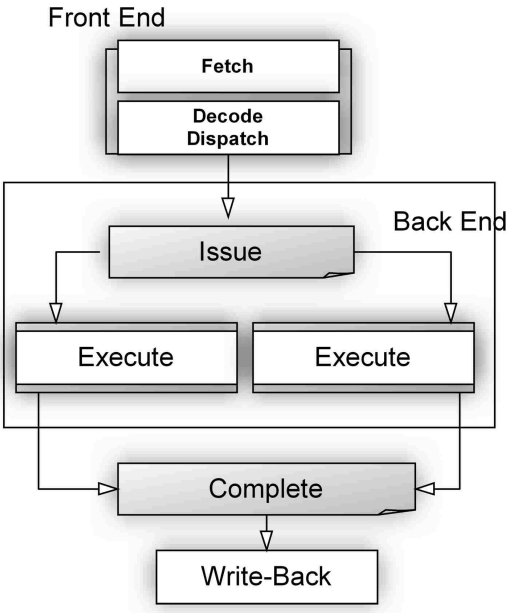


Figure A.7: Dynamic Scheduling with Issue Buffers [98]

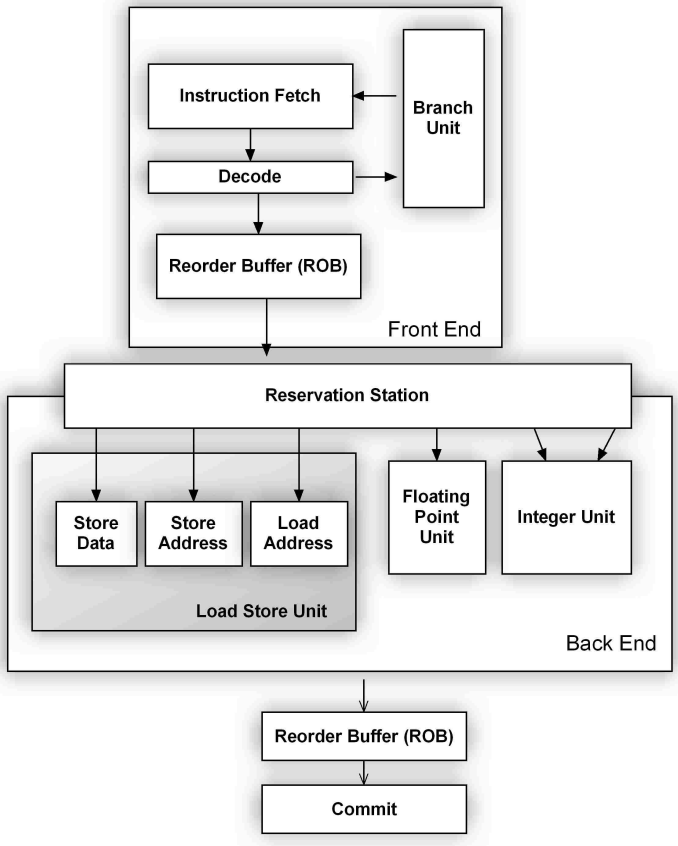


Figure A.8: The Pentium Pro [109, 98]

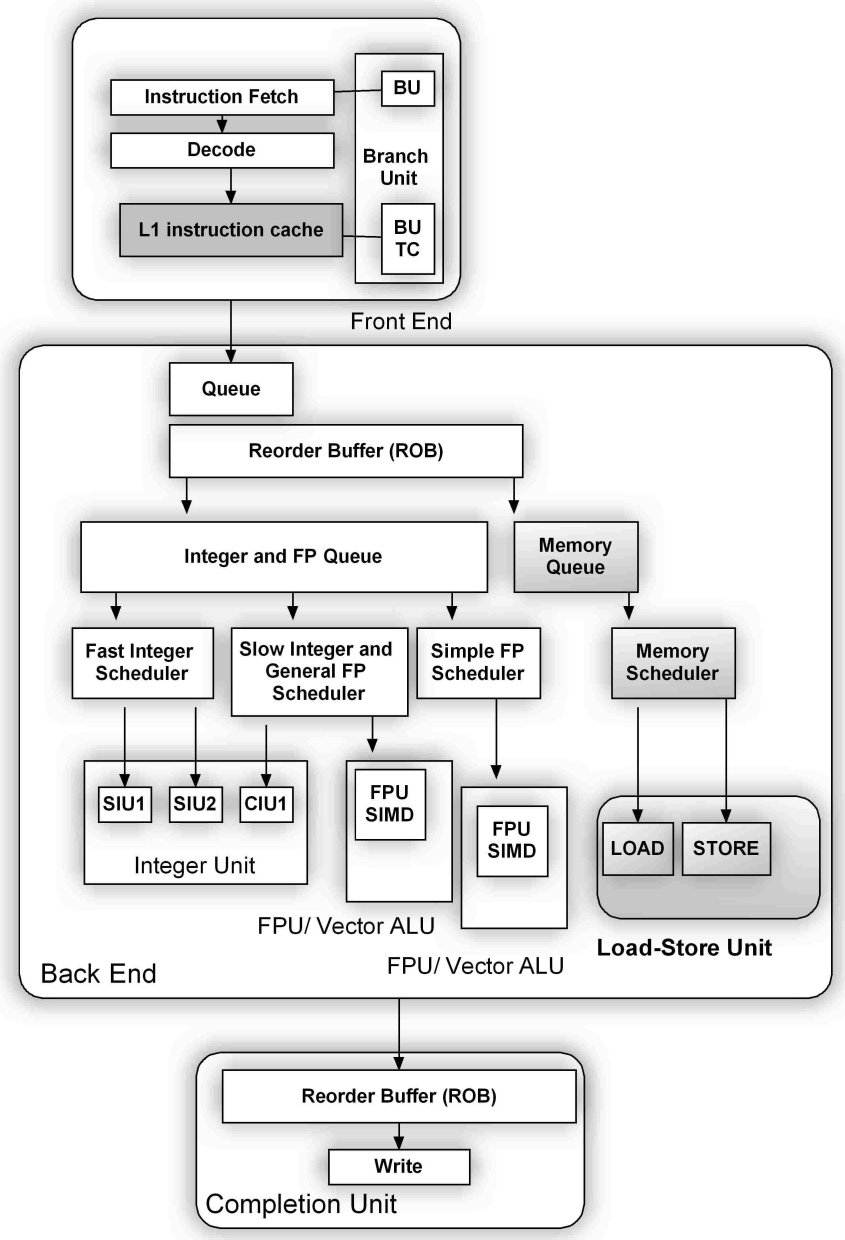


Figure A.9: Pentium 4: Basic Architecture [109, 98]

Appendix B
Network Exploration

Network exploration is a commonly used methodology in system and network administration for network inventory, host and service monitoring, and especially for security audits.

Moreover, today's security scanners are able to gather lots of useful information from remote network computers. Network discovery tools can determine name and version of offered services, presence of firewalls, type of packet filtering methods, filtered ports, device types, and even vendor of network cards in local area networks [25, 13].

We can basically classify network mapping into the following sub-categories [13]: (a) Host Discovery, (b) Port Scanning, (c) Version Detection, (d) OS Detection.

Host discovery is defined by detecting available hosts on a network. Most common approach is to use "ping" utility to determine whether a host is reachable or not across an IP network. Ping sends ICMP echo request packets and waits for a corresponding ICMP echo response from a target the computer machine, then calculates the round-trip time [24]. Operating system detection is accomplished by TCP/IP stack fingerprints [23]. TCP/IP flag settings may vary from one TCP stack implementation to another. Different implementations respond differently if incrorrect data is sent. Thus, a combination of data is sent to the system in order to discover its version according to the response [8]. In service detection, we first connect once an open port is found, and then, wait for the initial welcome banner because common services like FTP, SSH usually identify themselves in the welcome banner [18]. Basically, service scanners connect to the port and compare returned data in order to determine the program using that port.

B.1 Port Scanning

Port scanning is one basic technique used by system administrators to find out open ports in order to check the security of the computer. There are 65535 port numbers in TCP/IP, and they have been categorized in three ranges [8, 24, 11]: Well known ports (0-1023), Registered Ports (1024-49151), Dynamic/Private Ports (49152-65535).

A port scan to a specific port results in generally three states [8]. If host sends back a reply, it shows that the port is open, and a there is a service listening on that port. If reply says that the connection is denied, port status is closed and there is no service listening on that port. The third port status is filtered or blocked in which no response from the target host is sent back.

Port scanning has been classified as portscan, which searches a single host for open ports, and portsweep, which searches multiple hosts on a network to find a specific open port [11]. The simplest port scanning technique is TCP scan in which network function of the underlying operating system is used such that user does not require special privileges. In TCP scan, connect() system call returns success if port is open and listening, otherwise port is not accessible. However, this method will easily will be detected and logged by the system. Therefore, there are also many other techniques to proceed not be detected by auditing utilities.

Port scanning methodology has been classified as follows [24]: (1) TCP connect() scanning (2) TCP SYN (half-open) scanning, (3) TCP FIN (stealth) scanning, (4) TCP ftp proxy (bounce attach scanning), (5) SNY/FIN (fragmented IP packets) scanning, (6) UDP recvfrom() scanning, (7) UDP raw ICMP port unreachable scanning, (8) ICMP scanning (ping-sweep), (9) Reverse-ident scanning.

In TCP SYN-scan, which is usually known as half-open scan, we do not rely on provided network functions such that port scanner send a raw IP packet and gets the response. Instead of using TCP three-way handshake protocol, a SYN packet is sent pretending that a real connection will be opened [24, 8]. If port is open and listening, a SIN—ACK packet would be received. Otherwise, RST packet will be sent back from target host. When port scanner receives a SIN—ACK packet and determines port is open, it immediately closes the connection by sending a RST packet back [24, 8]. The advantage of half-open scan is that this access to the port is not logged by many systems.

Using raw IP packet gives full control in TCP stack, but root privileges are required to prepare custom IP packets. On the other hand, sophisticated scanning tools provide specialized network libraries and novel ways to use of raw IP packets [13].

Another stealth scanning technique is sending a FIN packet and attempting to close a connection which is not open. It is expected that operating system will generate an error and will reply back an RST packet if the target port is closed. If port is open, the sent packet will just be ignored and no response to the scanner indicates there is a service listening on that port. However, operating system may not behave as expected

and reply back always with RST packets to FIN requests [24, 8].

In fragmented packet scanning, TCP header is divided into several IP fragments [24, 8]. Since packet filters do not usually queue all IP fragments, port scanner can bypass the firewall. In ftp bounce attack scanning, port scanner take advantage of the vulnerability of ftp server which supports proxy ftp connections. Ftp servers accept a request which ask the server to open a data connection to a third party host on a given port [24]. Generated response code will help port scanner to identify the port and hide where scan attach is coming from.

In UDP scan, it is expected that system will response with ICMP port unreachable message if port is not open [24]. Non-privileged users cannot access this unreachable error message, but operating system can indirectly inform the user and scanner can determine port availability according to the difference in returned error messages [24]. ICMP echo packets are used to determine host availability. In ident scan, we take advantage of the ident service which gives information about the user that owns the service running on a specific port, so scanner can determine whether port is open and active [24].

B.2 Practice in Port Scanning

We have studied simple port scanners which support TCP connect() scan and SYN scan. In SYN scan, root privileges are required in order to use raw IP packets. Also, we have implemented a simple port scanner using TCP connect scanning technique. Our initial effort to write a network exploration function enabled Stork to detect whether remote host and service is available, before initiating the data transfer jobs. We have integrated host exploration feature in Stork scheduler transfer modules such that data placement jobs return with relevant error messages if availability of host and service is not justified.

```
$./stork.transfer.globus-url-copy ftp://virtdev/tmp/a gsiftp://virdtdev/tmp/b
Transferring from: ftp://virtdev/tmp/a to: gsiftp://virtdev/tmp/b with arguments:
$cat out.7478
Network error: can not resolve destination host - virdtdev !

$ ./stork.transfer.globus-url-copy file://virttest/tmp/a gsiftp://virtdev/tmp/b
Transferring from: file://virttest/tmp/a to: gsiftp://virtdev/tmp/b with arguments:
$cat out.7585
Network error: destination port virtdev:2811 is not open !
```

B.3 Practice with Nmap

Nmap is an open source tool for network mapping and security scanning. It uses raw IP packages in a ”novel way” for network exploration and service detection [13]. Nmap is a very powerful tool and it is extremely fast,and does not use the underlying network layer in the operating system. After its first version, many new features such as better detection algorithms, new scan types and supported protocols have been developed in the following versions [12, 13]. Besides many useful characteristics, Nmap provides XML output format to be easily interpreted.

A typical Nmap scan is shown below:

```
$ nmap  -A -p1-10000  -d  gridhub.cct.lsu.edu
.......
DNS resolution of 1 IPs took 0.00s. Mode: Async [#: 4, OK: 1, NX: 0, DR: 0, SF: 0, TR: 1, CN: 0]
........
PORT     STATE SERVICE     VERSION
22/tcp   open  ssh         OpenSSH 3.6.1p2 (protocol 2.0)
2222/tcp open  ssh         OpenSSH 3.9p1 NCSA_GSSAPI_3.5 GSI (protocol 1.99)
2811/tcp open  ftp         vsftpd or WU-FTPD
......
```

Nmap relies on the expected behavior of network services such that its performance was poor like other simpler scanners when target host was a well configured computer for security.

We have implemented a special functionality to be called inside Stork data placement scheduler, which is capable of using nmap techniques to resolve host name and to explore given hosts also return available services. This utility, which is basically calling Nmap objects, is dependent on many external Nmap network libraries to access raw Ethernet layer.

On the other hand, parsing XML output returned by Nmap utility is an easier way of implementation, and also Stork would be able to use all features of a port scanner. We have integrated Nmap functions into Stork data transfer modules in order to obtain speedy host discovery and port scanning.

We experimented network exploration and service detection techniques and used Nmap features inside data placement operations to resolve host, scan ports, and determine available services. More information about coding Stork modules for host discovery and port scanning, and also implementation details to integrate and use Nmap features for service detection and host discovery in Stork modules, can be found in [41].

Vita

Mehmet Balman received his bachelor of science and master of science degree in computer engineering from Bogazici University, Istanbul,Turkey. Currently, he has been studying for a doctoral degree in computer science at Louisiana State University. His research interests include high performance scientific computing, distributed data management and job scheduling in widely distributed systems.