Error Detection and Error Classification: Failure Awareness in Data Transfer Scheduling *

Mehmet Balman[1] and Tevfik Kosar[2]
[1] Computational Research Division, Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA
[2] Department of Computer Science, Louisiana State University
Baton Rouge, LA 70803, USA
Email: mbalman@lbl.gov kosar@lsu.edu

2010

Abstract

Data transfer in distributed environment is prone to frequent failures resulting from back-end system level problems, like connectivity failure which is technically untraceable by users. Error messages are not logged efficiently, and sometimes are not relevant/useful from users' point-of-view. Our study explores the possibility of an efficient error detection and reporting system for such environments. Prior knowledge about the environment and awareness of the actual reason behind a failure would enable higher level planners to make better and accurate decisions. 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. We investigate the applicability of early error detection and error classification techniques and propose an error reporting framework and a failure-aware data transfer life cycle to improve arrangement of data transfer operations and to enhance decision making of data transfer schedulers.

Keywords: error detection, error classification, network exploration, data movement between distributed repositories, scheduling bulk data transfer operations

Reference to this paper should be made as follows: Balman, M. and Kosar, T. (2010) 'Error detection and error classification: failure awareness in data transfer scheduling', *Int. J. Autonomic Computing*, Vol. 1, No. 4, pp.425-446

1 Introduction

Although latency and throughput are the main performance factors of data transfers (both in highly distributed and closely coupled environments), usability and efficiency of distributed data transfers also depend on some other aspects such as error detection and error reporting. Failure during data transfer in distributed environment is quite common. The major drawback in distributed data transfer is that the user sometimes is not aware of technical facts like the back-end 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 (Allcock et al., 2005; Allcock, 2003; GridFtp, 2006; FDT, 2009). Parallel data transfers, concurrent connections, and tuning network protocols such as setting TCP buffer are some of the techniques applied (Balman and Kosar, 2007a, 2009a; Yildirim et al., 2008; Dong and Akl, 2007). 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 consists of complex workflows where execution processes have data dependencies between each other. Scheduling and ordering of data movement tasks not only enhance the overall performance but also prevents failures and inefficient resource sharing (Balman and Kosar, 2007a,b; Kosar and Balman, 2008). Users usually do not have access to remote distributed resources and may not track the reason for 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 and data transfer schedulers 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 mitigate first is the lack of sufficient information to clarify the reasons for a failed data transfer. Our study has two main aspects: error detection and error classification. In error detection, we focus on making data transfer scheduler aware of whether destination host/service is available, and also making the scheduler able to select suitable data 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.

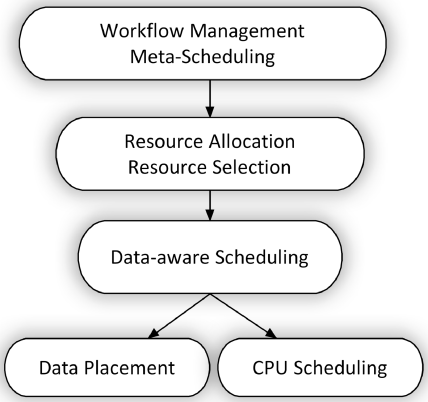
In this paper, we extend our initial study Balman and Kosar (2009b) on early error detection and classification, and present the architecture of an actual system for failure-aware data transfer scheduling. The outline is as follows. In Section 2, we first introduce the data-aware computing and data scheduling in distributed environments. In Section 3, we highlight some related work in failure-aware data scheduling. In Section 4, we propose a structural failure detection and error reporting mechanism. In Section 5, we introduce network exploration and we explain possible methodologies for using network exploration techniques in early error detection. In Section 6, we propose a failure-aware process cycle for data transfer operations, we analyze methods to keep track of operations while transfer is in progress, and we venture into how to use those methods in a data transfer scheduler. In Section 7, we elaborate on implementation issues and explain integration of error detection and classification into our data scheduler Stork (2009), (Kosar and Balman, 2008). We evaluate our methodology and give details about some real life experiments in Section 8. Finally, we put our conclusion, future work and open research problems in the area.

2 Scheduling Data Transfer Jobs

Data-aware computing and data scheduling approaches in distributed environments have been described in details in (Kosar and Balman, 2008; Balman and Kosar, 2007a). This section provides a brief summary about data-intensive computing for large scale applications. We encourage readers to look at Balman and Kosar (2007a) for data scheduling models and use cases in distributed environments. In addition, Balman (2008) gives more information about implementation issues and presents the structure of Stork (2009) data scheduler.

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.

Figure 1: Data-aware System Model

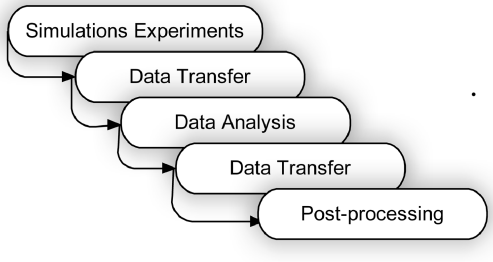


The SuperNova project in astronomy is producing terabytes of data per day, and a tremendous increase is expected in the volume of data in the next few years (Aldering et al, 2002). The LSST (Large Synoptic Survey Telescope) is scanning the sky for transient objects and producing more than ten terabytes of data per simulation (Tyson, 2002). 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 (CERN, 2007), processing of petabytes of experimental data remains the main problem affecting quality of the real-time decision making. In Kosar (2005), data handling issues of Blast (Altschul et al., 1990), a bioinformatics application, and CMS (2007), 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.

Large scale data-intensive 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 1 represents a detailed view of data-intensive system structure.

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.

Figure 2: Data Flow in Large-Scale Applications



One important feature is fault tolerant transfer of data objects. Storage servers may create problems due to too many concurrent write requests; data may be corrupted because of a faulty hardware; transfer can hang without any acknowledgement. We necessitate a scheduling mechanism which can collect information from separate sites and order data transfer jobs accordingly in order to enhance the overall system performance. We should minimize the possibility of failures and also handle faulty transfer in a transparent way.

3 Related Work

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 (Cieslak et al., 2008). 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 root cause of erroneous cases (Cieslak et al., 2008). 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 Thain and Livny (2002). 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 (Thain and Livny, 2002). An implicit error represents an invalid functionality. Explicit errors are due to an inability to accomplish the requested operation (Thain and Livny, 2002). 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 (Thain et al., 2003; CONDOR, 2008). We essentially concentrate on errors in data transfers for distributed large scale applications.

A fault tolerant middleware has been described in Kola et al. (2004) 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. The

methodology described in Kola et al. (2004) uses log information from the job scheduler and the data placement scheduler and takes the next action according to user policies. 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.

4 Structural Failure Detection and Error Reporting

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 to get any information about a failure.

Although, data transfer protocols 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 mal-functionality 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 not 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.

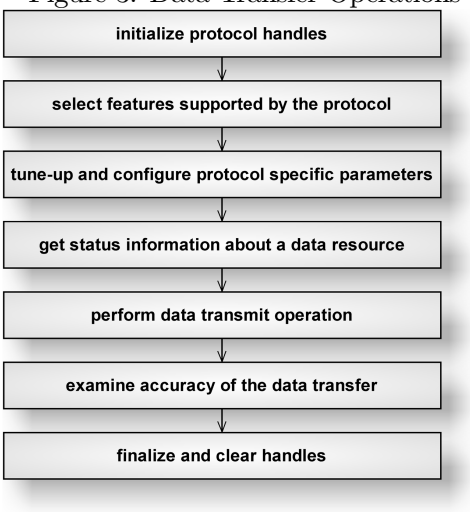
4.1 Framework for Error Classification

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 3.

In the initialization phase, all parameters are set and connection is established to have control over 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 3: Data Transfer Operations

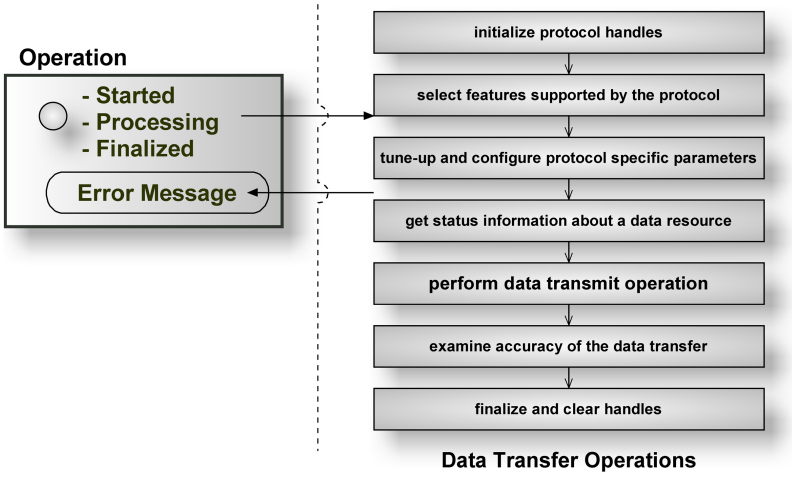


The main purpose of classifying data transfer operation in several categories is to better understand at which stage an error has occurred. File transfer protocols such as GridFtp (2008) generate error codes and error messages However, proposed error reporting framework will help both users and higher 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 order. First, status information of a file will be examined, and then in the later stages, checksum or size of a file will be requested. We initialize required transfer handles, activate modules specific to the protocol and prepare the system to connect and transmit information. A failure in the fist stages shows that either protocol is not supported or a proper connection can not be established. If failure happens during the parameter configuration phase, different set of tune-up options can be used to accomplish a successful transfer. User specific errors such as invalid file names, permission and authorization problems, or an attempt to get non-existing resource can be detected in the status phase.

As an example, a directory transfer operation can fail since used file transfer protocol is not supporting directory listing. In such a case, it will fail in the status phase before proceeding to the transmit phase. Therefore, we can provide a better logging facility which can be parsed and used by a higher level planner to get information in which stage operation failed. Besides, we can also understand in which point an error has occurred in each stage. In order to capture errors caused by network failures or mal-functionality in the protocol, we

Figure 4: Error Reporting Framework for Data Transfer Operations



keep state information in every phase. If we get an error after a file transfer operation has already been initiated and data transmission is started for processing, we treat the problem according to the fact that a problem may occurred in the network or remote site. Therefore, we define the operation object with three state types to keep track of status information between each phase. An operation can be in (a) start, (b) processing, and (c) end state. An error condition may have different meanings whether it is before or after processing state in the operation. Figure 4 shows the interaction of operations in the error reporting framework.

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 Error Detection with 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. On a computer network with the fact that services are not advertising themselves by a service discovery protocol, it is very much valuable to discover computers available on the network, and to determine what services are running. 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, operating system version of available hosts on the network, presence of firewalls, type of packet filtering methods, filtered ports, device types, and even vendor of network cards in local area networks (Host Detection,

2001; Nmap, 2008; Port Scanning, 1997). We experimented network exploration and service detection techniques and used Nmap features inside data movement operations to resolve host, scan predefined ports, and determine available services.

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

The following five layers have been proposed in Kola et al. (2004) 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 (2008) (Port Scanning, 1997), are able to handle this 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).

Network exploration puts extra overhead, but according to our first hand experiments, shown in Section 8, 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. Interaction of network probing applications with security appliances, such as IDS or Firewall, is an important concern since any such appliance will try to detect and block network scans However, we use limited set of exploration techniques to resolve host address and to explore given hosts also return available data transfer services. We emphasize that a very small set of network probing methods (i.e. some functionality in Nmap) and simple network exploration and discovery approaches are used for early error detection. Therefore, our integration only includes special functionalities such that transfer modules return with relevant error messages if availability of host and service is not justified.

In our recent study (Balman, 2007, 2008), we discuss our experiments with network scanner implementations and clarify applicability of those mechanisms in Stork data scheduler. Our proposed error reporting framework and a failure aware data transfer life cycle benefits from network exploration tools. In Appendix A, we have explained network exploration in details and give information about Nmap (2008) (Nmap Manual, 2008) network scanner tool.

5.1 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 enabling Stork (2009) data placement scheduler 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 !
```

5.2 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 (Nmap Manual, 2008). 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 (Nmap, 2008; Nmap Manual, 2008). 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
......
```

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 Stork error detection modules implementation details to integrate and use Nmap features for service detection and host discovery can be found in Balman (2007).

6 Data Transfer Life Cycle

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 Section 5, 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. 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 or by executing basic functions in the client interface of the protocol.

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. 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.

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.

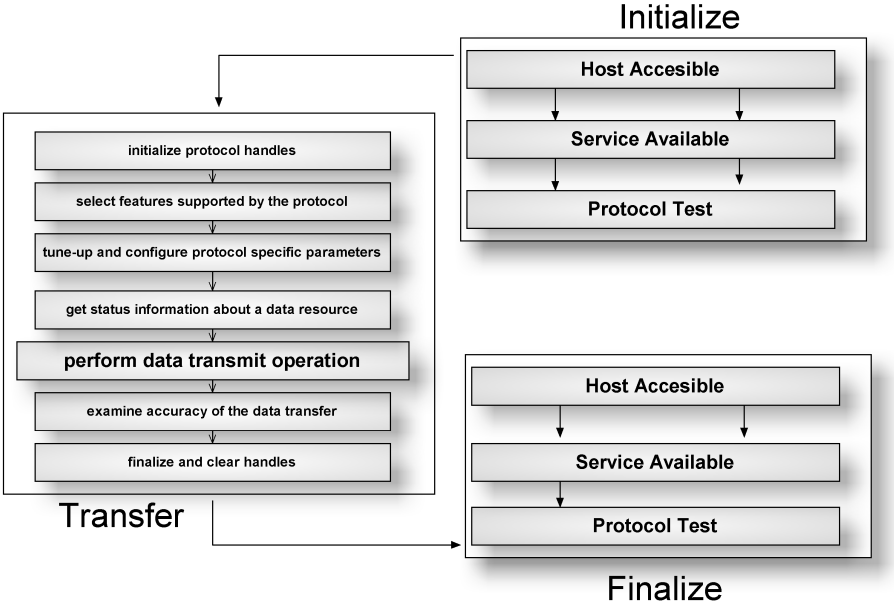
After testing the connection availability, we perform service detection techniques such that we ensure the requested service is running in the remote site. This step has twofolds; we simply detect a possible failure due to unavailable service in the target host, and we can use the information of other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

As it has been described in Section 4, 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.

We have prepared a testbed to detect possible erroneous cases in distributed data transfers and used GridFtp as our file transfer protocol. During our experiments with GridFtp (2008), 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

Figure 5: Data Transfer Progress Cycle



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, 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.

6.1 A Case Study with GridFtp Data Transfer Protocol

GridFtp client API (GridFtp, 2008; Allcock, 2003) 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 (Globus, 2008), 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 channel 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.

Our experiments, in which we are generating artificial errors for testing purpose, shows

that current data transfer protocol are not always able to generate adequate log information; therefore we also focus on tracing the transfer job and preparing the infrastructure to explore dynamic instrumentation while transfer is in progress.

The following gives a glimpse of the possible erroneous situation where artificial faults are injected by modifying the network settings of the test server.

```
$ globus-url-copy  -dbg -v -r
 gsiftp://dsl-turtle06.csc.lsu.edu/tmp/test/*.txt file:///tmp/1/ &
.......................
200 PBSZ=1048576
debug: sending command:
PASV
debug: response from gsiftp://dsl-turtle06.csc.lsu.edu/tmp/test:
227 Entering Passive Mode (130,39,225,167,213,123)
debug: sending command:
MLSD /tmp/test
..................... HANGS here

$strace -p 21298
..........
read(7, "632 FwMAAFDPNc7O+iHi7AWKmv2vJIi2"..., 100) = 100
........
fcntl64(8, F_GETFL)                     = 0x2 (flags O_RDWR)
fcntl64(8, F_SETFL, O_RDWR|O_NONBLOCK)  = 0
connect(8, {sa_family=AF_INET, sin_port=htons(54526),
sin_addr=inet_addr("130.39.225.167")}, 16) = -1 EINPROGRESS ..........
write(7, "ENC FwMAADDma01sEymyaFdU7edOLDZY"..., 78) = 78
select(9, [4 7], [], NULL, NULL
..................... HANGS here
```

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 time-out for data transfer operations. In the protocol test stage, if we cannot get response within this time-out limit, we can mark this remote service as faulty. In the following, we give more information on dealing with those type of erroneous cases by tracing data transfer operations.

6.2 Discussion on Dynamic Instrumentation: Tracing While Transfer in Progress

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. However, it is hard to trace if we have a problem while transfer is in progress. We have 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 remote data transfer server may not return back any data in the data channel due to some misconfiguration or some other server side problem. As it has been explained in the previous sub-section, 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 (Cantrill, 2007; Cantrill et al., 2004). 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 (Cantrill et al., 2004). 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. 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. There are providers for monitoring scheduling service, system calls, I/O devices, and I/O requests, IP, virtual memory, etc. (Cantrill et al., 2004).

Dtrace (2007) and similar tools have great importance in terms of dynamically tracing a running system. Many example scripts for performance monitoring are provided in Dtrace ToolKit (2007); moreover, there is an effort on developing Dtrace network providers for network observability (Cantrill et al., 2004). On the other hand, there are also alternative tools like SystemTap (2007) in which we are also able to track reads and writes on socket initiated by the process.

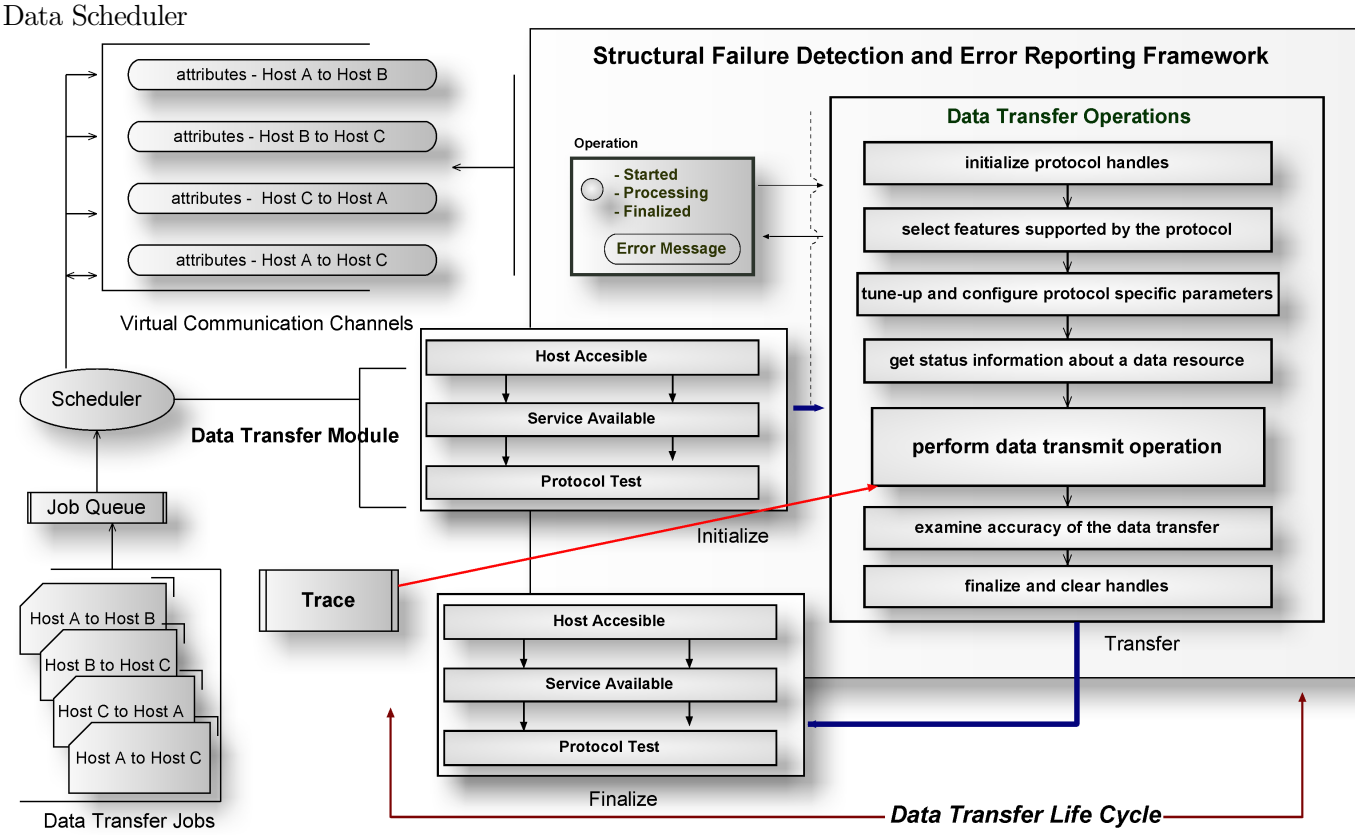
The asynchronous property of GridFtp protocol makes it hard to track an operation when transfer is in progress. We first initiate a transfer operation and register read/write functions with appropriate callback functions, and then, wait callback functions to be executed. We studied instrumentation techniques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

7 Integration into Data Transfer Scheduler

Scientific applications especially in several areas such as physics, biology, and astronomy have become more data intensive such that storage requirements went from giga- to peta-scale (Hey and Trefethen, 2003; Atlas, 2007; CMS, 2007; Holtman, 2001; Allcock et al., 2001b). Through the use of geographically distributed resources, organizations gain access to the resources

Figure 6: Integration of Structural Failure Detection and Error Reporting Framework into Data Scheduler



needed for their large-scale applications. On the other hand, We deal with a dynamic network layer where data movement middleware needs to adapt to the changing conditions in the environment. Furthermore, heterogeneous resource and different data access and security protocols are some other challenges. Complex middleware is required to orchestrate the use of these storage and network resources between collaborating parties, and to manage the end-to-end distribution of data. We have presented a data scheduler, Stork (2009), in order to organize data movement operations in collaborative peta-scale computing systems.

Stork data scheduler has a modular architecture such that data transfer operations are executed by external transfer modules (Kosar and Livny, 2004). We propose an abstract connection layer inside the data scheduler to keep track of network statistics. This enables us to utilize previous status information of failed operations.

For every source-destination pair encountered so far, the data scheduler updates network information such as host accessibility, available data transfer services. Whether an error condition has encountered, the status for the target host and service are updated. Therefore,

Table 1: Experiments with and without Network Exploration

(a) Overhead of Early Error Detection for Successful Transfers

		average transfer time	overhead of early detection
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)

(b) Benefit of Network Detection Feature for Failed 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)

the scheduler can use this information and delay data transfer jobs which are requesting transfer operations to or from a remote service that has been marked as faulty. The error reporting framework and tracing transfer operations are sub-processes besides the main server for better scheduling decisions.

The data transfer life cycle explained in this study is designed as the failure aware process cycle in Stork data placement scheduler. Figure 6 shows a brief summary about integration of error detection and classification model into the data 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.

8 Evaluation and Discussion

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 SCOOP (2008) project (Stamey et al., 2007) for hurricane Gustav simulations generated in 2008. Our test environment includes LONI (2007) 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 take average values of total transfer times to examine the overhead of network exploration. As can be seen in Table 1(a), overhead is ignorable; besides, early error detection module provides faster recognition of the network error, shown in Table 1(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.

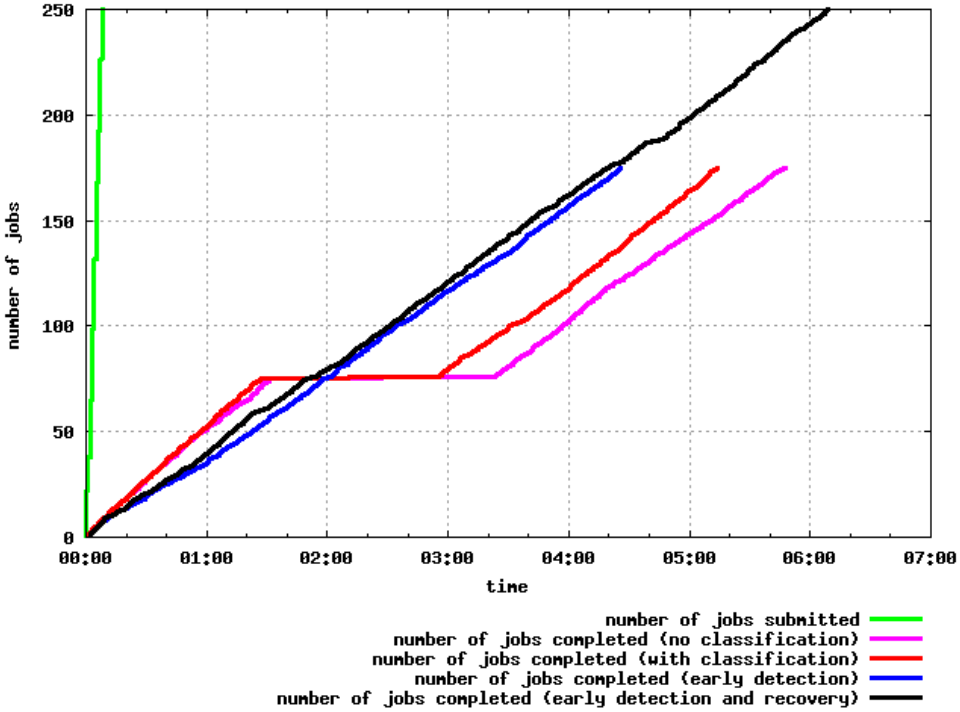
Next, we have experimented the impact of error detection and classification in data transfer scheduling. We used 250 data transfer jobs submitted to Stork scheduler and injected different types of errors into the system while the scheduler is performing given requests. Simply, we change the permission 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 7 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.

Stork, data placement scheduler, checks network connection and availability of the data transfer protocol. Early error detection has been implemented as a new feature inside Stork scheduler. The network exploration feature will also enable the scheduler to select alternative protocols among available data transfer services provided in a remote storage site. Moreover, we propose a generic framework such that error classification is not limited by GridFtp operations. We have also been testing our model with other data transfer protocols like iRods (SDSC, 2008).

The GridFtp transfer module in Stork is also able to verify the successful completion of the operation by controlling checksum of each file. Moreover, it can recover from a failed operation by restarting failed data transfer operations. The rescue file keeps track of failed and succeeded file transfer operations. In case of a retry from a failure, the scheduler informs the transfer module to recover and restart the transfer using the information from a rescue file created by the checkpoint-enabled transfer module.

Figure 7: Performance Effect of Error Detection and Classification in Scheduling Data Transfer Operations



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.

9 Conclusion and Future Work

Error detection and error classification have not been studied in details 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 experimented some of the features and implemented transfer modules to be used inside Stork, data placement scheduler. We have tested our system with 105,000 data transfer jobs for the movement of 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. Besides, we emphasize the need to explore dynamic instrumentation while transfer is in progress.

Acknowledgements

This project is supported by the National Science Foundation under award numbers CNS-0619843 (PetaShare) and EPS-0701491 (CyberTools), and by the Board of Regents, State of Louisiana, under Contract Numbers DOE/LEQSF (2004-07), NSF/LEQSF (2007-10)-CyberRII-01, and LEQSF(2007-12)-ENH-PKSFI-PRS-03, and in part supported by the Office of Science, U.S. Department of Energy, under contract no. DE-AC02-05CH11231.

References

DTrace (2007). DTrace Network Providers. Available online at http://opensolaris.org

Dtrace ToolKit (2007). Available online at http://opensolaris.org/os/community/dtrace

Nmap (2008). Available online at http://insecure.org/nmap

Nmap-Manual (2008). Nmap Reference Guide (Man Page). Available online at http://insecure.org/nmap/man

Vscan (2007). Service and Application Version Detection. Available online at http://insecure.org/nmap/vscan

Stork (2009). Stork Data Scheduler. Available online at http://www.cct.lsu.edu/ balman/stork/downloads.php.

SCOOP (2008). SURA Coastal Ocean Observing and Prediction (SCOOP) Program. Available online at http://scoop.sura.org

SystemTap (2007). Available online at http://sourceware.org/systemtap/

Finger-Prints (2008). TCP/IP stack FingerPrinting. From Wikipedia. Available online at http://en.wikipedia.org

Port-Scanning (1997). The Art of Port Scanning. Available online at http://insecure.org/nmap

Host Detection (2001). Advanced Host Detection Techniques To Validate Host Connectivity. Synnergy Networks. 2001

Aldering, G. and SNAP Collaboration (2002). Overview of the SuperNova/Acceleration Probe (SNAP). Available online at http://www.citebase.org/abstract?id=oai:arXiv.org:astro-ph/0209550

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

Allcock, B., Foster, I., Nefedova, V., Chervenak, A., Deelman, E., Kesselman, C., Lee, J., Sim, A., Shoshani, A., Drach, B., and Williams, D. (2001b). 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. ACM Press.

Allcock, W. (2003). GridFTP Protocol Specification. Global grid forum. GFD.20.

Allcock, W., Bresnahan, J., Kettimuthu, R., and Link, M. (2005). The globus striped GridFTP framework and server. In *Proceedings of the ACM/IEEE conference on Supercomputing*, page 54. IEEE Computer Society.

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

Atlas (2007). A Toroidal LHC ApparatuS Project (ATLAS). Available online at http://atlas.web.cern.ch

Balman, M. (2007). Using Network Exploration and Service Detection Techniques in Stork-Data Placement Scheduler. Center for Comp. & Technology, LSU http://csc.lsu.edu/~balman/technical.html.

Balman, M. (2008). Failure-awareness and dynamic adaptation in data scheduling. *M.S. Thesis, Louisiana State University.*

Balman, M. and Kosar, T. (2009a). Dynamic adaptation of parallelism level in data transfer scheduling. *International Workshop on Adaptive Systems in Heterogeneous Environments (ASHEs 2009), Fukuoka, Japan.*

Balman, M. and Kosar, T. (2009b). Early error detection and classification in data transfer scheduling. *Proceedings of International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC-2009), Fukuoka, Japan.*

Cantrill, B. (2007). Hidden in Plain Sight. ACM Queue 4 (1): 26-36.

Cantrill, B., Shapiro, M. W., and Leventhal, A. H. (2004). Dynamic Instrumentation of Production Systems. Proceedings of the 2004 USENIX Annual Technical Conference.

CERN (2007). The world's largest particle physics laboratory. European Organization for Nuclear Research. Available online at http://public.web.cern.ch

Cieslak, D., Chawla, N., and Thain, D. (September 2008). Troubleshooting Thousands of Jobs on Production Grids Using Data Mining Techniques. *IEEE Grid Computing.*

CMS (2007). The US Compact Muon Solenoid Project. Available online at http://cmsinfo.cern.ch/outreach

CONDOR (2008). Condor project. Available online at http://www.cs.wisc.edu/condor/.

Dong, F. and Akl, S. G. (2007). Two-phase computation and data scheduling algorithms for workflows in the grid. *Parallel Processing, International Conference on*, 0:66.

FDT (2009). Fast Data Transfer. Available online at http://monalisa.cern.ch/FDT

Globus-url-copy (2008) Globus Multi-protocol data movement. Available online at www.globus.org

GridFTP (2008) GridFTP Developer's Guide. Available online at http://www.globus.org

GridFTP (2006). Protocol Extensions to FTP for the Grid. Available online at http://www.globus.org/grid_software/data/gridftp.php

Hey, T. and Trefethen, A. (2003). The Data Deluge: an e-Science Perspective. *In F. Berman, G. C. Fox, & Anthony Hey (Eds.), Grid Computing: Making the Global Infrastructure a Reality. Chichester, UK: John Wiley & Sons, Ltd.*, pages 809–824.

Holtman, K. (July 2001). CMS data grid system overview and requirements. *CMS Note 2001/037,CERN*, 99.

Kola, G., Kosar, T., and Livny, M. (2004). Phoenix: Making Data-intensive Grid Applications Fault-tolerant. In *5th IEEE/ACM International Workshop on Grid Computing*.

Kosar, T. (2005). Data Placement in Widely Distributed Systems. Ph.D. Thesis, University of Wisconsin-Madison.

Kosar, T. and Balman, M. (2008). A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems, The International Journal of Grid Computing: Theory, Methods and Applications, Elsevier, 2008*, DOI: 10.1016/j.future.2008.09.006.

Kosar, T. and Livny, M. (March 2004). Stork: Making Data Placement a First Class Citizen in the Grid. In *International Conference on Distributed Computing Systems*.

LONI (2007). Louisiana Optical Network Initiative. Available online at http://www.loni.org

Mate, P. (2007) Internet Security. Available online at http://www.cs.wright.edu/ pmateti

Balman, M. and Kosar, T. (2007a). 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.

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

SDSC (2008). San Diego Supercomputer Center iRODS project. https://www.irods.org/.

Stamey, B., V.Wang, and Koterba, M. (August, 2007). Predicting the next storm surge flood. *Sea Technology*, pages 10–15.

Teo, L. (2007), Network Probes Explained: Understanding Port Scans and Ping Sweeps. available online at http://www.linuxjournal.com/article/4234

Thain, D. and Livny, M. (2002). Error scope on a computational grid: Theory and practice. In *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC'02)*, pages 199–208. IEEE Computer Society.

Thain, D., Tannenbaum, T., and Linvy, M. (2003). Grid Computing: Making the Global Infrastructure a Reality. In *Condor and the Grid*, number ISBN:0-470-85319-0, pages 299–336. John Wiley.

Tyson, J. A. (2002). Large Synoptic Survey Telescope: Overview. In Tyson, J. A. and Wolff, S., 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.

Yildirim, E., Balman, M., and Kosar, T. (2008). 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. ACM.

A Appendix: 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 (Host Detection, 2001; Nmap Manual, 2008).

We can basically classify network mapping into the following sub-categories (Nmap Manual, 2008): (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 (Port Scanning, 1997). Operating system detection is accomplished by TCP/IP stack Finger-Printing (2008). TCP/IP flag settings may vary from one TCP stack implementation to another. Different implementations respond differently if incorrect data is sent. Thus, a combination of data is sent to the system in order to discover its version according to the response (Mate, 2007). 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 (Vscan, 2007). Basically, service scanners connect to the port and compare returned data in order to determine the program using that port.

A.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 (Mate, 2007; Port Scanning, 1997; Teo, 2007): 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 (Mate, 2007). 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 (Teo, 2007). 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 (Port Scanning, 1997): (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 (Port Scanning, 1997; Mate, 2007). 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 (Port Scanning, 1997; Mate, 2007). 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 (Nmap Manual, 2008).

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 (Port Scanning, 1997; Mate, 2007).

In fragmented packet scanning, TCP header is divided into several IP fragments (Mate, 2007; Port Scanning, 1997). 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 (Port Scanning, 1997). 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 (Port Scanning, 1997). 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 (Port Scanning, 1997). 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 (Port Scanning, 1997).