Method for Task Migration in Grid Environments*

Stephen Frechette and D.R. Avresky
Network Computing Lab, Electrical and Computer Engineering Department
Northeastern University
Boston, MA USA
{sfrechet,avresky}@ece.neu.edu

Abstract

This paper presents a method, infrastructure, and prototype that enables adaptive application task migration among a Grid environment. Our infrastructure automatically reconfigures distributed applications in response to network performance failures and denial of service (DoS) attacks. Through the use of performance monitoring software we enable network connection failover and automatic application task migration within a heterogeneous distributed computing environment. Our system distinguishes itself from other available adaptive computing frameworks because it is wholly composed of open source Grid-enabled components capable of both transparent and dynamic selection of message passing transports based on resource performance.

1. Introduction

Our framework monitors the performance of a diverse set of network connections and initiates application task migration in response to anomalies. We develop a sensor network with integrated effectors, which recognizes certain performance failures and follows a reconfiguration plan. Additionally, our framework enables applications distributed across a wide-area network of standard Ethernet connections to also utilize any high speed System Area Network (SAN) connection. Unlike previous task migration infrastructures, our software is wholly composed of Grid-enabled components encompassed within a C library, which automatically selects a message passing transport medium based on performance measurements.

Our Task Management C Library, and Self-Monitoring system, follows the Open Grid Service Architecture (OGSA) [4]. Our contribution is a novel message passing layer and a unique framework for the migration and management of application tasks. Network connections automatically reconfigure, and application tasks migrate, in response to network performance failures, resource consuming DoS attacks, or legitimate resource loads not attributed to the distributed application. The design and implementation of an open source, Grid-enabled, adaptive computing environment is described.

The remainder of the paper is organized as follows. The next subsection notes other application task migration infrastructures. Section 2 details the many components that compose the framework and gives an overview of the methodologies. Section 3 details the setup of experiments, which demonstrate the beneficial use of adaptive computing in the presence of network performance failures and DoS attacks. The results, presented in Section 4, demonstrate an improvement in the performance of a distributed application, which employs our adaptive computing framework and automatically reconfigures the environment in response to network performance failures and DoS attacks. Future work is presented in Section 5. The findings and conclusion are given in Section 6. Lastly, the Appendix contains in depth details about the sensors and the reporting mechanisms of the framework.

1.1. Related Work

Currently, there are a few Message Passing Interface (MPI) library based approaches that institute adaptive distributed computing environments. CoCheck MPI [3] is the first MPI implementation that uses the Condor library [28] for check-pointing and task migration. Our framework differs greatly from CoCheck MPI, in that it utilizes both SAN and wide-area network connections in addition to monitoring the network traffic via Grid-enabled tools. Other MPI implementations capable of task migration include FT-MPI [1], and MPICH-V [2]. The following packages represent current generation adaptive computing systems.

The GrADS system [8], which is currently evolving, is

0 This work was supported by the U.S. National Science Foundation under grant CCR-0004515

a performance oriented migration framework for the Grid. Both GrADS and the HARNESS framework [15] envision a library of Grid aware components for adaptive Grid computing. The GrADS system employs an application manager that creates performance contracts, which are used as initiators for application task migration [23]. The Application Level Scheduling (AppLeS) system [11] implements adaptive distributed computing, although, unlike other infrastructures it is not encompassed within a library. An attractive feature of AppLeS is its ability to employ various scheduling heuristics. This use of historical performance data as an aid for scheduling is described in [9], [18]. A fully functional and scalable Grid environment monitor is described in [16]. The AutoMate system [21] builds on Grid middleware and follows the OGSA. The core services detailed in OGSA entail the security and management of information, resources, and data.

2. Structural Overview

The interaction of our components, which enable the automatic reconfiguration of an application distributed across a heterogeneity of network connections, is illustrated in Fig. 1. Our framework, accessible via C Library functions, enables distributed applications to pass messages through both a wide-area network and a high-speed SAN. Through the creation of a SAN application level message passing service, a single application distributed across the Grid can seamlessly utilize 10 Gb/s InfiniBand SAN connections in conjunction with wide-area network connections. Although there currently exists an MPI implementation for InfiniBand, our framework differs in that it is capable of anomaly detection and enables an InfiniBand SAN connection to failover to a standard Ethernet connection.

The framework facilitates the use of high-speed SAN connections by applications distributed across a wide-area network. Our Task Management ANSI C Library creates and manages application tasks, which enable the developer to program an adaptive distributed application at a high level, thus the programmer is not over-burdened by all the complexities necessary for application task migration. Our framework improves the dependability of distributed applications in the presence of DoS attacks and network performance failures.

The distributed applications targeted for improvement by this paper use the standard MPI 1.1 library convention. We employ the Grid-enabled MPICH-G2 library [6], [7], which is an extension of the Argonne MPICH implementation of the standardized Message Passing Interface (MPI) version 1.1 [19]. This extension enables the use of services provided by the Globus Toolkit [29] for authentication, authorization, resource allocation, staging of executables, and startup/collective operations. The MPICH-G2 MPI implementation enables applications distributed across a Grid environment to use an MPI library.

2.1. Message Passing Service

An application level service is created to provide a message passing mechanism within the SAN via high-speed intracluster network links. This architecture reflects a modularization of communications mechanisms within the SAN network. This service provides a standard interface for an MPICH-G2 MPI program to seamlessly communicate with a limited set of MPI functions through a 4X InfiniBand network connection via IP over InfiniBand [24].

The message passing service responds to the following MPI functions: MPIV_Send and MPIV_Recv. The minor difference between these functions and the standard MPI 1.1 MPI_Send and MPI_Recv is the addition of pointers to the $node\_structure$ structures of the source and destination nodes. The $node\_structure$ is a C structure that contains the information necessary for task migration, $node\_structure$ structure is declared in Fig. 2 and read in Fig. 3.

The Task Management C Library enables the MPIV_Send and MPIV_Recv functions to pass messages through high-speed SAN network connections. The MPI 1.1 standard contains an MPI profiling convention; the purpose of the interface is to implement a profiling tool and additionally to internetwork multiple MPI implementations [26]. Through the use of our MPI functions, with the prefix MPIV, we enable the seamless use of multiple communication libraries by a single distributed application.

This SAN message passing service handles the MPIV_Send calls and immediately sends the message to the destination's Global Access to Secondary Storage

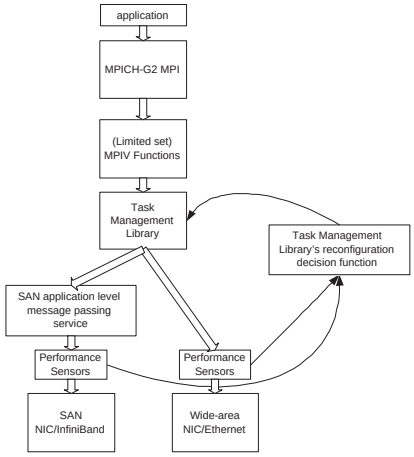


Figure 1. Message passing layer overview.

(GASS) server [12], which is included in the Globus Toolkit. The GASS server acts as a secure file server and is employed for message passing within the SAN, the Appendix contains more details about our use of the GASS server. The corresponding MPIV_Recv function polls the local machine for the sent message. The service may use various system level message passing mechanisms, such as either the InfiniBand Socket Direct Protocol [25], which takes full advantage of Remote Direct Memory Access (RDMA), or IP over InfiniBand. The Socket Direct Protocol exhibits a 2.7x higher throughput compare to the IP over InfiniBand protocol [20]. The wide-area message passing service and the control signals of the distributed application are passed via the open source Grid-enabled MPICH-G2 MPI library. Through the use of the MPICH-G2 MPI library our framework supports all MPI v1.1 functions for communication over the wide-area network.

2.2. Sensors

The Grid-enabled Ganglia monitor system [5] serves as the network monitoring system. Passive sensors are used to monitor both the SAN and the wide-area network. Ganglia monitors the CPU load, memory usage, disk usage, number of open TCP/IP connections, as well as over fifteen other statistics. Over 500 deployments of the Ganglia monitoring system are used by clusters throughout the world.

A localized network-centric approach is used to detect DoS attacks. The throughput of each computational node is determined by the number of bytes per second of application data sent over the given link by the node. The outgoing application level throughput of each computational node is measured passively at every call of the MPIV_Send function. This passive method of measuring performance statistics, and determining application behavior, is employed by various MPI profiling tools [23]. When the distributed application reports the throughput to Ganglia, if the throughput is below a predetermined threshold, then the performance degradation is detected within 30 seconds.

The sensors employed in this paper detects anomalies in Grid environments. Our monitoring system is scalable, accurate, timely, flexible, and incurs low overhead, these are important attributed for a computational Grid's monitoring system [22]. Our closed-loop controller, composed of sensors and effectors, is detailed in the Appendix.

2.3. Task Management Library

The Task Management Library integrates the sensors and effectors, which are described in the next subsection. Additionally, in order to support task migration, the Task Management Library provides a data structure for a virtual process topology. Each virtual process encapsulates and executes a certain application task, thus the terms virtual process and application task are used synonymously. The pseudo code of the library is shown in Figures 2 and 3.

```
Task Management Library Function Declarations:

typedef struct
{
int Interest_Group[MAX_NUMBER_OF_NODES];
    int InfiniBand_GroupA;  #1 if INFB conn., 0 otherwise
    int Global_ID;
    char Machine;
    char[30] GASS_URL;
    int job_description;
} node_structure;

MPIV_Send(void *buf, int count, MPI_Datatype datatype
          ,node_structure *source_pnode, node_structure *dest_pnode
          ,int message_id, int tag, MPI_Comm comm)

MPIV_Recv(void *buf, int count, MPI_Datatype datatype
          ,node_structure *source_pnode, node_structure *dest_pnode
          ,int message_id, int tag, MPI_Comm comm, MPI_Status *status)

Reconfiguration_decision(IN char *Global_status_file,
          IN int *Performance_contract, OUT int *XYZ_score)

Virtual_topology_reconfiguration(IN int real_node,IN int virtual_node,
   IN/OUT node_structure *nodes)

Virtual Topology Reconfiguration Function:

Virtual_topology_reconfiguration(new_node, old_node, nodes)
#MIGRATE old_node to new_node, i.e. put old_node to sleep and have
#new_node take its place.
{
nodes[virtual_node].Global_ID = real_node

#replace the occurrences of old_node in all .input and .output
#arrays with new_node
#package_reconfiguration_info contains new Global_ID
#(global specific) and #InfiniBand_GroupA (node specific).
foreach (X -> nodes[new_node].Interest_Group)
    FILL_recv_reconfig_node_info_variable(package_reconfiguration
                                          _info[X], X, nodes);

#tell affected processes to reconfigure
foreach (X -> nodes[new_node].Interest_Group)
    MPIV_Send(packaged_reconfiguration_info[X], SIZE_OF_UPDATE
              ,MPI_INT, X, RECONFIGURATION_TAG, MPI_COMM_WORLD);

unaffacted_nodes = queue_sub(all_nodes - affected_nodes)
#tell other unaffected processes to continue
foreach (X -> unaffacted_nodes)
    MPIV_Send(for_null_ie_continue,SIZE_OF_UPDATE, MPI_INT
              ,unaffected_nodes[X], CONTINUE_TAG, MPI_COMM_WORLD);
}
```

Figure 2. Declarations of Task Management Library and the virtual topology reconfiguration function.

All application tasks are locally aware, i.e., they only have knowledge of the application tasks with which they directly interact. All local knowledge is contained in the $Interest\_Group$ array shown in Fig. 2. In order to avoid deadlock during reconfiguration, a timeout will be included in the reconfiguration function in future versions of our library. Currently, even during a resource consuming DoS attack, the victim is required to eventually respond to a reconfiguration request.

2.4. Effectors

The Task Management Library contains $Leaders$ that enable task migration. The $Leader$ process is responsible for the execution of the $Reconfiguration\ decision$ function, shown in Fig. 2. The $Performance\ contract$ array contains a list of point-to-point virtual pro-

```
Measurement and Reporting Functionality

MPIV_Send(void *buf, int count, MPI_Datatype datatype, node_structure
         *source_pnode, node_structure *dest_pnode, int message_id
         , int tag, MPI_Comm comm)
{
byte_size = count
destination = (*dest_node). Global_ID
#Measure elapsed time for an MPI_Send.
If  ((*dest_node). InfiniBand_GroupA AND (*source_node).InfiniBand_GroupA)
    Fwrite(temp.bin, buf, byte_size)
    Time0 = gettimeofday()
    gass_copy(temp.bin, *dest_node. GASS_URL)
    elapsed = gettimeofday() - Time0
End
Else  #using MPICH-G2 TCP/IP connection
    Time0 = gettimeofday()
MPI_Send(buf, byte_size, MPI_INT,  destination, CONTINUE_TAG, MPI_COMM_WORLD);
    elapsed = gettimeofday() - Time0
End
byte_rate = byte_size / elapsed #byte_rate for a byte_size copy to gmetad
exec(/usr/bin/gmetric --name net_Source_location  --value byte_rate --type
     double --units bytes/sec)
}

MPIV_Recv(void *buf, int count, MPI_Datatype datatype, node_structure
          *source_pnode, node_structure *dest_pnode, int message_id
          , int tag, MPI_Comm comm, MPI_Status *status)
{
byte_size = count
destination = (*dest_pnode). Global_ID
#Measure elapsed time for an MPI_Send.
If  ((*dest_node). InfiniBand_GroupA AND (*source_node).InfiniBand_GroupA)
    Timeout = 300
    Time0 = gettimeepoch()
    For (;;)
        Time1 = gettimeepoch()
        Sleep( 1 second)
        successful_read = fread(temp.bin, buf, byte_size)
        If (((Time1 - Time0) > Timeout ) || successful_read) #Timout
           Break
    End
End
Else
MPI_Recv((int*) buf, byte_size, datatype, (*source_pnode).Global_ID
 , MPI_ANY_TAG, MPI_COMM_WORLD, status);
}
```

Figure 3. MPIV_Send and MPIV_Recv functions within the Task Management Library.

cess connections and their corresponding throughput threshold. The $Global\_status$ file reports the throughput of the connections within the virtual topologies, when a measured throughput falls below the threshold listed in the $Performance\ contract$ array, then the tasks which execute on the nodes that receive messages from the affected links are selected for migration. The $Reconfiguration\ decision$ function decides upon a reconfiguration based on information in the $Global\_status$ file and the $Performance\ contract$ array. The $Performance\ contract$ array serves as a contract and when it is violated the task is selected for migration by the $Reconfiguration\ decision$ function and the task subsequently migrates.

In our simple scheme whenever a $Performance\ contract$ is broken an application task is selected for migration, and the task migrates to the computational node that receives the largest volume of that task's data. The $Performance\ contracts$ are created during the execution of the distributed application under normal load conditions without any DoS attacks. An increase in traffic due to a DoS, or a spike in resource usage, negatively affects the available bandwidth, CPU time, memory, or some combination of thereof. Our system reacts to any surge in non-application load, it does not attempt to distinguish between DoS attacks and load spikes caused by legitimate traffic.

This paper does not address the decision of when it is best to migrate an application task, and ignores the potential problem of the creation of a clusters of application tasks that all migrate to the same resource, resulting in heavy loads which may cascade through the physical network. When the $Performance\ contract$ is broken we simply migrate an application task to the computational node that receives the largest volume of that task's data. The problem of determining when a task should be migrated off a node, taking into account all the scheduling and resource availability factors, is studied in more detail in book edited by one of the authors [30]. Due to the NP-Completeness of the scheduling of a parallel program represented by a weighted directed acyclic graph (DAG) to a set of heterogeneous processors [32], an optimal solution to this problem cannot be calculated in polynomial time for every instance, and we do not address this scheduling problem.

2.5. Task Migration

The virtual processes are created within the data structure of the Task Management Library. The MPI version 1.1 standard does not contain the functionality to spawn additional MPI processes after application startup, therefore it is required that spare processes are created at application startup. At application startup the $Leader$ process maps virtual processes to the physical topology. During application runtime the virtual processes can be dynamically migrated within the physical topology, in response to anomalies.

The $Leader$ process contains the Task Management Library's $Reconfiguration\ decision$ function and the globally aware virtual topology structure, which is composed of an array of $node\_structures$. This array of $node\_structures$ may reside in any machine within the distributed system, and our framework could allow for its migration. Currently, our framework does not have the functionality to migrate the $Leader$ process. A virtual topology consisting of virtual processes (1,2,3,4), is shown in Fig. 4, the remainder of the processes are spare and denoted by an e.

Figure 4 illustrates the migration of virtual process 2, i.e., application task 2, from machine X to machine Z. The code that executes an application task does not actually migrate from machine X to machine Z. An example of the migration of virtual processes is detailed as follows: once migration is decided on, a virtual process is signaled to become inactive on machine X and a spare process is signaled to become active on Machine Z, then configuration data is subsequently sent to the newly activated spare process. The spare MPI process, denoted by an e in Fig. 4, is in an idle

state until reconfigured and assigned a virtual process by the $Leader$. In response to either a network performance failure, or DoS attack, a spare process e in machine Z is assigned virtual process 2.

Machine Y and Z are connected by both an InfiniBand SAN network connection and a Standard Ethernet connection. The $Leader$ ensures that the nodes affected by the task migration contain valid and updated configuration data. All the nodes in the $Interest\_Group$ array, seen in Fig. 2, are synchronized before reconfiguration.

When the signal is sent for a real process to become active six items are included in the signal. These items are listed in the $node\_structure$ of Fig. 2. The items in the $node\_structure$ include the following information: virtual process ID, the virtual process IDs of the neighbors found in the $Interest\_Group$ array, whether or not the SAN can be accessed and if so, the URL of the GASS server.

3. Experimental Setup

We inject a network performance failure and launch a DoS attack upon a computational node, which participates in the execution of a distributed application. The experimental data presented in Section 4 is generated using three Intel Xeon/2.4 GHz dual processor machines and one Intel PentiumIII/1 GHz dual processor machine, all with Linux 2.4 kernels.

3.1. Setup of a Network Performance Failure (Standard Ethernet only)

The first experiment consists of an application task migration, in response to a network performance failure, within a Grid environment that exclusively consists of standard Ethernet network connections. Figure 5 illustrates a physical topology and an application task migration in response to either a network performance failure or a DoS attack. In this experiment, an existing distributed data compression application [27] is slightly altered to utilize our Task Management Library, then the application is monitored during a network performance failure.

In this paper, we define a network performance failure as a significant decrease of the available point-to-point throughput between machines within the distributed computing system, that persists for more than 30 seconds. An injected network performance failure is illustrated in Fig. 5. The network performance failure is created by an 80 MB copy from a remote machine to machine X, resulting in a degradation of Machine X's available throughput.

3.2. Setup of a DoS Attack (Standard Ethernet only)

The term DoS attack is defined as any attack, in which the attacker attempts to cause a system's resources to become too busy to respond to other clients. DoS attacks work on the premise that the Internet and computers are composed of a limited set of vulnerable resources such as bandwidth, disk space, and CPU cycles. There are many different types of DoS attacks, one is a bandwidth consumption attack. A bandwidth consumption attack entails the posting or request of large amounts of data to or from the victim computer via packet streaming. This type of attack is successful if it degrades the network throughput, thus negatively affecting other clients attempting access.

The chosen form of DoS attack for this experiment is a bandwidth consuming DoS attack, selected for its ease of use and controllability. The Apachebench [14] and an Apache http server are employed to implement the attack. One remote machine initiates a DoS attack against a target computer by issuing an Apachebench command, which opens 200 concurrent connections to the http server running on the target computer. Each concurrent connection requested 8 MB of data from the target computer. Control of the consumed bandwidth is achieved through the adjustment of both the number of concurrent requests that are generated from the remote machine and the size of the requests.

3.3. Setup of a Grid-Enabled SAN Network (InfiniBand and Standard Ethernet)

The last experiment consists of a DoS attack on a computational Grid environment that consists of both 10 Gb/s InfiniBand and Standard Ethernet network connections, which is illustrated in Fig. 4. This experiment does not measure the performance of an application, but rather measures the volume of messages sent from application task 1 to application task 4. These messages are sent over both InfiniBand and standard Ethernet network connections, they originate at task 1 and are passed along by task 2 and task 3, then arrive at the destination, task 4.

In this experiment a single machine in our computational Grid environment is the victim of a DoS attack. Two of the machines, Y and Z, are connected to a 10 Gb/s 4X InfiniBand switch, in addition to the standard Ethernet connection. The InfiniBand SAN network connection uses Mellonox drivers and the IP over InfiniBand protocol to pass messages via writes to the Globus Toolkit's GASS server [12]. Figure 4 is an illustration of the physical topology of a small cluster consisting of Grid-enabled components, which our framework is implemented upon. The system performance, in terms of the number of MPI

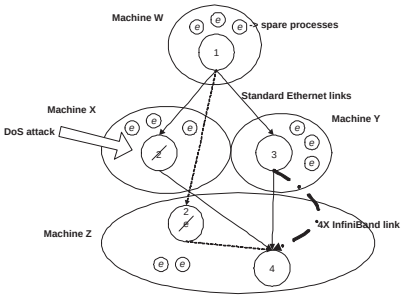


Figure 4. A task migration demonstration upon a sample virtual to physical topology mapping, task 2 migrates to machine Z.

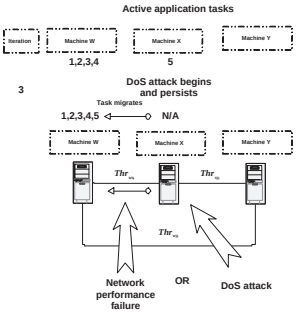


Figure 5. A task migration demonstration upon a sample virtual to physical topology mapping, task 5 migrates to machine W.

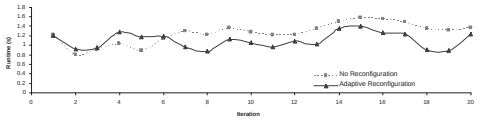
messages received by task 4, is experimentally determined and presented in the next section.

4. Experimental Results

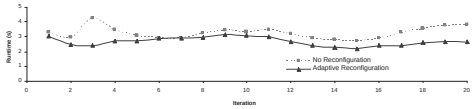
4.1. Network Performance Failure

The setup for this experiment is detailed in subsection 3.1. The runtimes of two trials of a data compression application were measured. The first trial exhibited a compression ratio of 0% and the second a compression ratio of 60%. For this particular data compression application the compression ratio corresponds to the compressibility of the input data. Both data compression applications received 524 kB of data as an input. The different compression ratios exhibited reflect the different profiles of the input data, and affect the amount of data passed between the nodes.

The runtime of the two trials are illustrated in Fig. 6a and Fig. 6b respectively. A moderate improvement in the performance of the application was measured. While the distributed application experiences a network performance failure, which begins at iteration 3, a 17% decrease in runtime was observed through the employment of



a) 0% compression ratio was observed



b) 60% compression ratio was observed

Figure 6. Runtime of a data compression algorithm, a network performance failure began at iteration 6 in figure a) and iteration 3 in figure b).

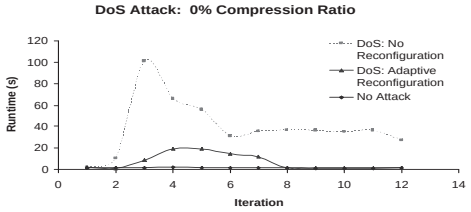
adaptive task migration compared with task migration disabled, as seen in Fig. 6a. The reconfiguration was performed automatically by the Task Management Library and initiated by a network performance failure.

Figure 6b illustrates the runtime of trial 2, which exhibits a compression ratio of 60%. A 16% decrease in runtime was realized through the use of the task migration framework, compared to a run without task migration enabled.

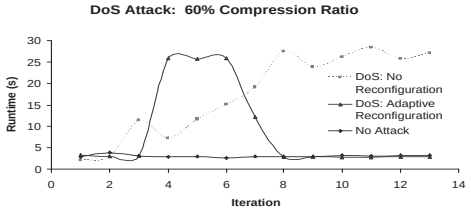
4.2. DoS Attack

The setup for this experiment is detailed in subsection 3.2. The performance of the distributed data compression application was monitored during a DoS attack. The distributed application consists of four application tasks capable of migration among the SAN network. Figures 7a and 7b plot the execution time of the data compression distributed application corresponding to the different compression ratios. The reconfiguration transition period, lasting from iteration 3 to 8, is seen in Figures 7a and 7b. During this period the DoS attack is detected by the $Reconfiguration\ decision$ function, which periodically analyzes the $Performance\ contract$ array and the $Global\_status$ file. Next, the Task Management Library sends the signals for an application task to migrate. Through the use of adaptive reconfiguration the data compression algorithm with a compression ratio of 0% exhibited an improvement in the runtime by more than 2,600%, compared to the case without task migration. A 900% improvement in the execution time of the data compression algorithm with a 60% compression ratio was also measured.

In the last experiment a DoS attack is implemented against machine X, seen in Fig. 4. The setup for the last experiment is described in subsection 3.3. The distributed application experiment utilizes both InfiniBand and standard Ethernet network connections, seen in Fig. 4,



a) 0% compression ratio was observed



b) 60% compression ratio was observed

Figure 7. DoS attack at iteration 3, shown in Fig. 5, upon one of the computational nodes that took part in the distributed data compression algorithm.

between machine Y and machine Z. This attack causes task 2 to migrate from machine X to machine Z. Figure 8 plots the cumulative volume of MPI messages sent to task 4, of Fig. 4, with and without adaptive reconfiguration enabled. The DoS attack begins at time 1. Compared to the volume of task completion messages received with adaptive reconfiguration disabled, and after an initial settling time, a 20% improvement in the volume of messages received in the given time period was measured through the use of adaptive reconfiguration during a DoS attack. During the first 10 seconds of execution, the trial with adaptive reconfiguration enabled receives more task completion messages than the trial in which no attack is committed. This is due to the fact that during the adaptive reconfiguration task 2 migrates to the same machine where task 4 resides. As a result of this, messages will not traverse through machine X. Therefore, more task completion messages will be received before the system settles to a steady state, at time 14.

5. Future Work

The prototype implemented in this paper contains only four processes, which are synchronized by the $Leader$ processes before reconfiguration. Future work could include distributed synchronization that allows for multiple groups or multiple $Leader$ processes within the distributed application to concurrently reconfigure. The algorithm selected for distributed synchronization must

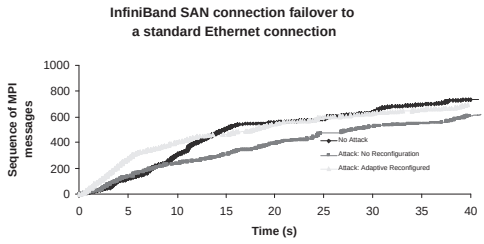


Figure 8. Sequence of MPI messages received by task 4, seen in Fig. 4, from both InfiniBand and standard Ethernet network connections.

exhibit freedom from both deadlock and starvation, in addition to fairness, and fault tolerance. Additionally, the prototype only reports the throughput of data passed via MPIV_Send. Future work could include the passive monitoring of all the application's traffic; subsequently the $Reconfiguration\ decision$ function would decide if the performance contract is violated.

With the additional functionality of a fault tolerant version of MPI, FT-MPI [1], a distributed application could dynamically spawn and kill processes to adjust as a networks unexpectedly scales up and down. One of the autonomic computing paradigms of self-management could be met, within an a distributed application which uses the FT-MPI library, through the employment of our Task Management Library and monitoring system.

6. Conclusion

Our framework improves the utilization of high value resources within a Grid environment by enabling the seamless use of high-speed SAN network connections by an application distributed and managed across a wide-area network. The open source Grid-enabled framework enables application task migration in response to DoS attacks and performance failures, thus improving the dependability of applications distributed across a heterogeneous Grid environment. Additionally, our infrastructure enables a distributed application to seamlessly pass certain MPI messages over both a high-speed SAN and standard Ethernet network connections. Our results show an improvement in the dependability of distributed applications that are dependent upon resources vulnerable to DoS attacks and performance failures.

References

[1] G. Fagg and J. Dongarra, "FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World," *Euro PVM/MPI User's Group Meeting 2000, Springer-Verlag, Berlin, Germany, pp. 346-354, 2000.*

[2] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov, "MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes," *IEEE Proc. of SuperComputing, 2002*.

[3] G. Stellner, "CoCheck: Checkpointing and Process Migration for MPI," *IEEE Proc. of SuperComputing, 2002*.

[4] J. Nick, I. Foster, C. Kesselman and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure," *IEEE Proc. of SuperComputing, 2002*.

[5] M. Massie, B. Chun, and D. Culler, (2003 Nov.). The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. [Online]. Available: http://ganglia.sourceforge.net/

[6] N. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," *J. of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 551-563, May 2003.

[7] N.T. Karonis and B. Toonen, (2003, Nov.). MPICH-G2. [Online]. Available: http://www.hpclab.niu.edu/mpi

[8] R. Fowler, (2003, Aug.). GrADS: Grid Application Development Software Project. [Online]. Available: http://hipersoft.cs.rice.edu/grads/publications_reports.htm

[9] K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, and O. Sievert, "Toward a Framework for Preparing and Executing Adaptive Grid Programs," *Proc. of NSF Next Generation Systems Program Workshop*, April 2002.

[10] MDS Functionality in GT3. (2003 Jun.). [Online]. Available: http://www.globus.org/ogsa/releases/final/docs/ infosvcs/MDS.html

[11] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive Computing on the Grid using AppLeS," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 4, pp. 369-382, Apr. 2003.

[12] Global Access to Secondary Storage (GASS). (2003 Nov.). [Online]. Available: http://www-fp.globus.org/gass/

[13] C. Dumitrescu, M. Wilde, and I. Foster, (2002 Nov.). Hierarchic Ganglia - a Distributed Monitoring Tool for Grid Environments. Univ. Chicago, IL. [Online]. Available: http://people.cs.uchicago.edu/ cldumitr/research/Ganglia-2.3.x.doc

[14] The Apache Software Foundation, (2003 Nov.). Apache Http Server Project. [Online]. Available: http://httpd.apache.org/

[15] S. Vadhiyar and J. Dongarra, "A Performance Oriented Migration Framework for Grid," *Proc. Third Int'l Symp. on Cluster Comp. and the Grid*, 2003, pp. 130-137.

[16] T.C. Ferreto, C.A.F. de Rose, and L. de Rose, "RVision: An Open and High Configurable Tool for Cluster Monitoring," *IEEE/ACM Int'l Symp. on Cluster Computing and the Grid*, 2002, pp. 75-82.

[17] F. Sacerdoti, M. Katz, M. Massie, and D. Culler, "Wide Area Cluster Monitoring with Ganglia," *IEEE Int'l Conf. on Cluster Computing*, 2003, pp. 289-299.

[18] L. Yang, J.M. Schopf, and I. Foster, "Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments," *IEEE Proc. of SuperComputing*, Nov. 2003.

[19] MPI: A Message-Passing Interface Standard, (1995 Jun.). Message Passing Interface Forum. Univ. Tennessee, TN. [Online]. Available: http://www-unix.mcs.anl.gov/mpi/

[20] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda, (2003 Oct.). Sockets Direct Procotol over InfiniBand in Clusters: Is it Beneficial? Ohio State University, OH, Tech. Rep. OSU-CISRC-10/03-TR54 [Online]. Available: http://nowlab.cis.ohio-state.edu/projects/mpi-iba/publication/balaji-sdp-dc-tr.pdf

[21] M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, and M. Parashar, "AutoMate: Enabling Autonomic Grid Applications", *Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS)*, 2003, pp. 48-59.

[22] P. Stelling, I. Foster, C. Kesselman, C.Lee, and G. von Laszewski, "A Fault Detection Service for Wide Area Distributed Computations", *Proc. of seventh IEEE Symp. on High Performance Distributed Computing*, 1998, pp. 268-278.

[23] S. Vadhiyar and J. Dongarra, "Self Adaptivity in Grid Computing". To appear in: *Special issue of Concurrency: Practice and Experience on Grid Performance*, 2003.

[24] IP over InfiniBand (IPoIB). (2002 May.). [Online]. Available: http://infiniband.sourceforge.net/NW/IPoIB/index.htm

[25] Sockets Direct Protocol. (2002 May.). [Online]. Available: http://infiniband.sourceforge.net/NW/SDP/index.htm

[26] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference Volume 1 - The MPI Core*, second ed. Cambridge, MA: MIT Press, 1998, pp. 388-390.

[27] D. Avresky, N. Natchev, and V. Shurbanov, "Dynamic Reconfiguration in High-Speed Computer Networks", *Proc. of IEEE Symp. on Cluster Computing*, 2001.

[28] D. Epema, M. Livny, R. van Dantzig, X. Evers, and Jim Pruyne, "A Worldwide Flock of Condors : Load Sharing among Workstation Clusters," *J. on Future Generations of Computer Systems*, Vol. 12, 1996.

[29] The Globus Toolkit. (2004 Aug.). [Online]. Available: http://www.globus.org

[30] J. Trotter and T.A. Varvarigou, "Reconfiguring Multiprocessor Systems while Minimizing Disturbance," *Fault-Tolerant Parallel and Distributed Systems*, D. Pradhan and D.R. Avresky, eds., Washington, DC: IEEE Computer Society Press, pp.122-131. 1995.

[31] A.G. Ganek, C.P. Hilkner, J.W. Sweitzer, B. Miller, and J.L. Hellerstein, "The Response to IT Complexity: Autonomic Computing," *IEEE Int'l Symp. on Network Computing and Applications*, 2004, pp. 151-157.

[32] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, NY, NY: W.vH. Freeman & Company, 1979.

Appendix

DESCRIPTION OF THE CLOSED-LOOP SENSOR NETWORK: Each computational node executes Ganglia's $gmond$ (B) daemon, Fig. 9, which monitors and broadcasts the statistics. Each cluster contains one or more $gmetad$ (D) daemons, Fig. 9, which aggregates the statistics from all nodes within the cluster. This information is stored in an open XML-based format, which allows for standard access. The Globus Toolkit's Monitoring and Discovery Service (MDS) [10] contains the URL that points to these statistics. At every MPIV_Send call Ganglia's $gmetric$ (C) daemon, which can report any arbitrary statistic, reports the measured throughput to Ganglia's $gmetad$ (D) daemon.

The globally aware $gmetad$ daemon pulls and summarizes the aggregate data from each cluster's gmetad (D) daemon and stores the data in an XML-based round robin database (rrd) file format. This aggregate data could be accessed by the Ganglia web front-end (E), which displays summary statistics of the state of each cluster over time. If implemented in a large-scale network the aggregate data could reside within a hierarchy of servers [13]. If a hierarchy of $gmetad$ (D) daemons, which hold a summary of the descendant's data, is deployed then Ganglia exhibits a linear increase of network bandwidth as additional computational nodes are added to the environment [17]; therefore the monitoring system is scalable. It is worth noting that the $gmond$ (B) daemon consumes less then 1% of the CPU's cycles [17]. The overhead was measured as the difference in runtime of the data compression application with the following components alternately enabled and disabled: the $gmond$ (B) and $gmetad$ (D) Ganglia monitoring daemons and the Task Management Library's sensors and effectors. A 1% overhead was incurred upon the distributed application when adaptive reconfiguration effectors and the network sensors were enabled.

The GASS server resides on the machines connected via link A, as shown in Fig. 9.

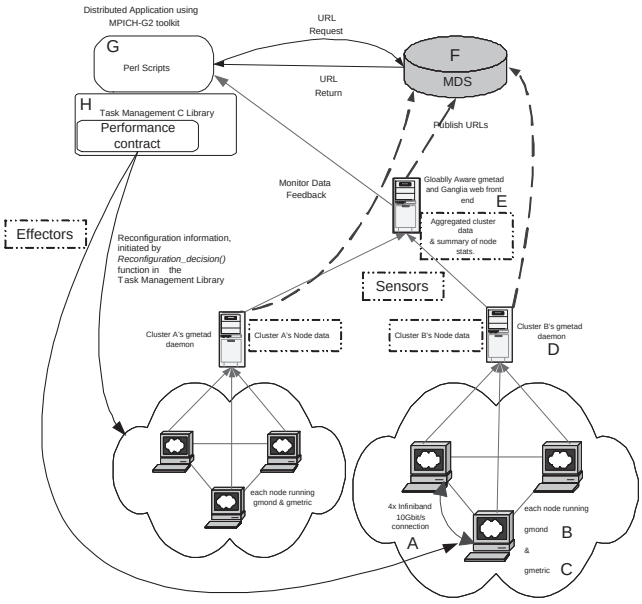


Figure 9. Framework of the adaptive distributed computing system.