Designing, Building, and Modeling Maneuverable Applications within Shared Computing Resources

William Clay Moody
Clemson University

Designing, Building, and Modeling Maneuverable Applications within Shared Computing Resources

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

by
William Clay Moody
May 2015

Abstract

Extending the military principle of maneuver into war-fighting domain of cyberspace, academic and military researchers have produced many theoretical and strategic works, though few have focused on researching actual applications and systems that apply this principle. We present our research in designing, building and modeling maneuverable applications in order to gain the system advantages of resource provisioning, application optimization, and cybersecurity improvement. We have coined the phrase "Maneuverable Applications" to be defined as distributed and parallel application that take advantage of the modification, relocation, addition or removal of computing resources, giving the perception of movement. Our work with maneuverable applications has been within shared computing resources, such as the Clemson University Palmetto cluster, where multiple users share access and time to a collection of inter-networked computers and servers. In this dissertation, we describe our implementation and analytic modeling of environments and systems to maneuver computational nodes, network capabilities, and security enhancements for overcoming challenges to a cyberspace platform. Specifically we describe our work to create a system to provision a big data computational resource within academic environments. We also present a computing testbed built to allow researchers to study network optimizations of data centers. We discuss our Petri Net model of an adaptable system, which increases its cybersecurity posture in the face of varying levels of threat from malicious actors. Lastly, we present work and investigation into integrating these technologies into a prototype resource manager for maneuverable applications and validating our model using this implementation.

Dedication

This dissertation is dedicated to my wife, daughter, and two sons who have provided their unconditional love through my entire graduate education.

Acknowledgments

I truly believe that you cannot do life alone, and that core value also definitely applies to research. You cannot do research alone. As such, I want express sincere appreciation to so many people and acknowledge their contributions to my dissertation. Additionally, many have been influential in my Army career that has empowered me to be able to have this wonderful opportunity to purse my doctorate and teach at West Point.

I first want to thank my Lord and Savior Jesus Christ who has provided me with immeasurably more than I could have asked for or imagined (Ephesians 3:20). All of my achievements are truly a blessing from Him.

Next, I want to thank my advisor, Dr. Amy Apon, for taking a chance and allowing me to be part of the laboratory and her research group. She has allowed me to follow my own passions with the research and to address the growing need to operationalize cyberspace. She has truly taught me how to become a researcher, author, and better student. Additionally, she has opened up her classroom to me for many semesters and allowed me to improve by teaching style and skills. I will forever be grateful for her guidance and mentorship.

My other committee members have also provided me valuable feedback and guidance. Dr. Pradip Srimani gave me such confidence building positive feedback. He recognized in me an untapped analytical thinking skill and research ability that encouraged me to excel in building system and conducting experiments for analysis. Dr. K.-C. Wang introduced me to the amazing potential of software-defined networking and opened up many doors in the U.S. Ignite and Clemson Computing and Information Technology communities. Finally, Dr. Hongxin Hu provided a valuable security researcher perspective to help me expand my interest in moving target defense.

Additionally, I want to thank my co-authors for their contributions in getting published works out for the research community to become familiar with our work. Listed in alphabetical order, Jason Anderson, Triiip Bowen, Judson Dressler, Eddie Duffy, Jason Koepke, and Linh Ngo have provided valuable feedback and assistance in this journey.

I also want to thank my mentors in the Army who have led and guided me towards the continually pursuit of knowledge and technical proficiency. Major General John Davis, Colonel (Retired) Ed Drose, Colonel (Retired) Jeff Schilling, Colonel John Norris, and Lieutenant Colonel (Retired) Patrick Mangin have been amazing mentors and leaders that have shaped my career. Additionally, I would have never reached this point without the help and encouragement of Colonel Greg Conti of the U.S. Army Cyber Institute and the United States Military Academy. Our months providing Cyber Support in Iraq opened up so many doors and I am so exciting to be joining you and the amazing faculty at West Point. I also want to thank my peers throughout my career who held me accountable and encouraged me while setting the pace for continued education, Vic Deekens, Stephen Hamilton, T.J. O'Connor, and Benjamin Sangster have been friends and colleagues that have made me a better officer and gentlemen.

I also cannot forget to acknowledge the love and support I have received from my church family at NewSpring Church. My current and former home group members, my JeepUp group, and my Campus Safety team have been the rock and foundation for my family and me during our time in Anderson.

Lastly, I have to thank my family. To my parents, Larry and Rhonda Moody, my in-laws, C.J. and Mimbee Ray: I love you all so much and thank for the close support during our time back home. It has been great to have you all so close and available for these three years. I wish my late father-in-law, Dr. Broughton Baker, Jr, DMD, could be here for another Clemson Graduation. To my three children, Belton, William, and Jeffrey: Thanks for your continued love and putting up with our many moves through the years. I know you will love New York. And finally, to my better half, my awesome wife, Kemper: You are my life. I love you so very much. This could not have been possible without knowing that you had my back. The past 17 years have been amazing, you are my best friend and the best is yet to come.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In 2010, William J. Lynn III, former United States Deputy Secretary of Defense, publicly released the details of a previously classified incident of the spread of malicious computer code across the U.S. military's unclassified and classified computer networks [59]. This malware was the product of a foreign intelligence service and was spread through the unauthorized movement of USB based flash drives between computing systems with different levels of classification. The efforts of the Department of Defense to counter the attack was known as Operation Buckshot Yankee and was managed at the highest levels of the Department due to the sensitivity and urgency. The highly visible release of information to the public highlighted the scope of daily cyber attacks and intrusions faced by the Pentagon and its military. This incident emphasized the need for improved operations and defense in the newest warfighting domain.

Since the mid 1940s, warfighting has been restricted to the traditional domains of land, sea, and air. Recent technological advances have led to adding space and cyberspace to the list of operational domains. With the expansion of domains to conduct combat operations, traditional military doctrine, technology, and tactics must be reevaluated or expanded to provide consistency across the entire military spectrum. This challenge has presented a plethora of opportunities for strategic debate, theoretical analysis, and technological research.

Many traditional military topics have been expanded and applied into the cyberspace domain [72]. These include the concept of situational awareness [32], key terrain [76], and defense in depth [87]. A basic military concept that has received extensive theoretical review is *maneuver* [16,35,83]. Though empowered by this theoretical research, there has been limited work in building systems that exhibit the characteristics and behaviors of maneuver [21, 22, 42].

Maneuver is one of the U.S. Army's nine Principles of War [17]. Maneuver includes the application of combat power to maintain an advantage over the enemy. This flexible and dynamic employment of resources ensures success by keeping adversarial conditions imbalanced and thus reduces failures, compromises, and vulnerabilities. The non-military use of the word *maneuver* describes an action that is not random or without purpose, but one that is clever or skillful. In both environments the word maneuver implies deliberate movement and actions taken to achieve a specific purpose.

Distributed and parallel applications allow the execution of complex computations that previously were deemed impractical. Many recent technological advances have contributed to the widespread growth of distributed computing, namely multi-core processors, multi-processor nodes, high speed networks, improved storage technologies, and virtualization. Regional and national researchers have combined funding and resources to build expansive, large-scale shared computing clusters to allow more efficient usage of power, space, and cooling to multiple user groups. These shared computing resources have become the standard high performance computational platforms that annually appear on the list of the most powerful commercially available computer systems known [9]. Even with these tremendous achievements, the need for continued progress in resource availability, optimization, and security exists.

Our research is motivated by the increased interest in further defining and abstracting the concept of military maneuver into the cyberspace domain along with the lack of implementation of maneuverability into actual systems. As such, our research is focused on designing, building, and modeling maneuverable application. We have coined the phrase "maneuverable applications" as being distributed and parallel systems and programs that take advantage of the modification, relocation, addition or removal of computing resources within the application, giving the perception of movement. These resources can be computation, network, or storage or can be the applications themselves. These actions are deliberate, purposeful and meant to achieve an advantage over adversarial conditions. Towards this end goal, this document presents our dissertation proposal.

1.1 Problem Statement

We present the following research question for this dissertation: How can the military principle of maneuver be applied to computing applications in order to gain system advantages over potential adversaries or limiting conditions?

1.2 Research Approach and Contributions

The research approach for this work is along four main avenues. The first approach is to study resource provisioning achieved through maneuver by the designing of an application used by students, researchers, and administrators. This system provides a big data processing platform in academic environments. The second approach is to explore the use of maneuver for application optimization by the building of a reconfigurable network testbed for studying the impact of physical and logical topology on datacenter performance. The third is to investigate maneuver to achieve a more secure computing system through the use of Petri Net models and simulation. The final approach is the synergistic integration of the previous technologies, models and results into a prototype maneuver resource manager along with the creation and discussion of best design practices for system providers who desire to produce applications that can exist in a maneuverable context.

1.2.1 Contributions

Our contributions in the area of resource provisioning include those documented in our paper to *CLUSTER 2013* [65]. We present a platform that enables the integration of Hadoop into an existing large scale academic computational infrastructure. This work features automated scheduling mechanisms that facilitate an arbitrarily long run time of MapReduce applications within the administrative constraints of a shared environment and with minimal interactions required from system administrators. We present experimental results demonstrating that maneuvers can occur as frequently as the average length of the reduce tasks of the executing MapReduce jobs, with only modest impact to performance.

The contributions of our work in application optimization include those in our work at *DIDC 2014* [63]. We introduce the design and prototype development of a datacenter testbed with a reprogrammable network topology. The testbed includes a Virtual Topology Engine that builds virtual network topologies over physical links with Software-defined networking flows and a Flow Network Evaluation system to generate a network congestion estimation score. We also present the design and development of a Hadoop shuffle traffic simulator designed to place realistic load on a datacenter network. Our work features experimental results to show that placement of computation racks within a datacenter topology can have significant impact of the Hadoop shuffle traffic completion time.

Our work with modeling security improvement through maneuver contributes to the understanding of security in distributed applications. We present the results in our work to *TRUSTCOL 2014* [62] and submitted to *EAI Endorsed Transactions on Collaborative Computing* [64]. We introduce a Stochastic Petri Net (SPN) model of a defensive maneuver cyber platform utilizing moving target defense and deceptive defense tactics. We analyze our SPN model to understand the trade-offs between security and operations in defensive maneuver cyber platform. We present a robust evaluation in the presence of an attacker and provides rules of thumb for deploying the system to meet certain operational and security objectives. We present recommendations for how to use and extend our current SPN model to build prototype systems implementing moving target and deceptive defense in parallel and distributed applications.

Our work with the Prototype Resource Maneuver Manager, Fortified, is a culmination of the previous works to build an operational maneuverable application basde on our security modeling. Our contributions include a fully functioning prototype software suite to deploy maneuverable Hadoop clusters into distributed environments with individual nodes maneuvering between operational, idle, and deceptive modes of operation. Additional, Fortified provides a discrete event simulator version of the Defensive Maneuver Cyber Platform that allows long running simulations to provide statistical analysis of the behaviors of the system. We present a validation of the mathematical probabilities of survival when targeted by an modeled attacker and a verification of the security enhancements provided by moving-target and deceptive defense mechanisms.

1.2.2 Thesis Statement

By applying the concept of military maneuver to parallel and distributed computing, systems can be built to provide resource provisioning, application optimization, and security improvement.

1.2.3 Thesis Organization

This thesis is organized as follows: Chapter 2 covers the background information, Chapter 3 covers our work using maneuver for resource provisioning with the Job Uninterrupted Maneuverable MapReduce Platform. Chapter 4 discusses our work in applying maneuver to provide application optimization with the Flow Optimized Route Configuration Engine. Chapter 5 covers our modeling of security improvement through maneuver with the Defensive Maneuver Cyber Platform. Chapter 6 introduces our prototype resource maneuver manager, Fortified and its use as modeling validation tool. Finally, conclusions are presented in Chapter 7.

Chapter 2

Background

In this chapter, we provide background information on topics in military cyberspace, distributed and parallel technologies, and system modeling tools. This overview establishes foundation for understanding the completed work and the proposed work for the dissertation research.

2.1 Military Cyberspace Doctrine

The growth of reliance on information systems and the advancement of technology has created much research and discussion on military doctrine concerning cyberspace operations. We discuss some of the theoretical and scientific studies in this section.

Alexander makes a case for warfighting in cyberspace and provides an overview of the then current state of the U.S. Department of Defense's cyber task organization [15]. He discusses Cyberspace as a war-fighting domain and contrasts cyber operations with information operations and electronic warfare. He argues that the U.S. must move quickly in determining the way forward doctrinally and technically in order to stay the world leader in this field. He compares the current environment to that of the period between World Wars when the world was struggling with how to apply airpower in combat operations.

Many authors have focused on understanding how the principles of war are similar and different with cyberspace and the traditional domains of warfare. Liles, et al., analyze the conventions of land warfare in light of cyber warfare and its effect on command and control, information flows and the adversary and friendly commanders [58]. He argues there is a significant disconnect between conventional warfare and cyberwar, especially in the laws defining what is considered an attack and intelligence gathering. Farmer presents a thorough overview of the history of the principles of war and the foundational documents for each service in the U.S. military [35]. Additionally, he applies cyber warfare context to all nine principles to understand if they are still valid in today's environment. His conclusion is they still hold true today and even offer insight into the driving nature and future enhancement of cyber operations. Parks and Duggan find the that traditional principles of war still apply to cyberspace operations but believe they need expanding upon and thus have created their own "cyberwarfare principles." They provide examples of where each traditional principle applies to cyberspace and they introduce their new principles with examples based on their red-teaming experiences. Kallbert and Thuraisingham challenge universities to increase the productivity of cybersecurity research in both doctrine, humanities and technology in an interdisciplinary approach [53]. They identify and analyze many hindrances to conducting successful research in cyber defense and operations along with present results of a survey from recognized centers of excellence in cyber research. Their results show a focus on information assurance and legal issues, but few universities researching the development of offensive and defensive capabilities.

2.1.1 Cyberspace Maneuver

Though much interest in "cyberspace maneuver" exists, there has been no official definition by the United States Department of Defense. The U.S. Military defines maneuver as "A movement to place ships, aircraft, or land forces in a position of advantage over the enemy" [29]. The U.S. Army's definition is to "place the enemy in a disadvantageous position through the flexible application of combat power" [17].

As far back as 1997, military strategists have been speculating on maneuver in an information domain. In his Marine Corps War College thesis [83], Major Singh of the Australian Army introduces the concept of maneuver in cyberspace operations. In the early days of the World Wide Web and prior to the existence of military doctrine of cyberwar, Singh outlined ramifications of such a conflict, justifications for exploration of national policies, and design of potential offensive and defensive capabilities to operate in this space.

Applegate presents the most extensive analysis of maneuver in cyberspace operations [16]. He provides motivation for cyberspace maneuver, defines maneuver in cyberspace and provides its characteristics. His definition is "the application of force to capture, disrupt, deny, degrade, destroy or manipulate computing and information resources in order to achieve a position of advantage in respect to competitors." Additionally he provides basic forms of offensive and defensive cyber maneuver.

Farmer explains cyber maneuver challenges center around the constantly evolving nature of the environment "due to technical adjustments including the addition, removal, replacement, or re-configuration of components or networking protocols." He demonstrates maneuver in the cyberspace domain with examples of the 2009 undersea fiber cable cuts and the Iranian election atrocities Twitter campaign. These examples show that cyberspace avenues can quickly be changed through nature or government intervention, and that creative applications of resources can help recover from these predicaments.

2.1.2 Moving Target Defense

Moving target defense (MTD) has recently received a great deal of attention [33, 34, 50, 51, 80, 94]. In [51], the authors address basic research challenges and how MTD can be deployed using asymmetric cost techniques that are advantageous for the defenders and disadvantageous for the attackers. In [34], the authors explore the effectiveness of MTD protection mechanisms. A model for dynamic diversity defense is proposed. In [50], an OpenFlow Random Host Mutation (OF-RHM) scheme is introduced to use OpenFlow to efficiently assign different addresses to hosts and protect against internal and external scanning. In [33], the authors develop a Moving Target IPv6 Defense (MT6D) that leverages the immense address space of IPv6 to hide and rotate IPv6 assignments by implementing MT6D tunneled packets. The two goals of MT6D are to maintain user privacy and protecting against targeted network attacks. A similar functionality in the form of a Linux hypervisor is provided by [93]. In [80], the authors describe an approach of injecting a artificial diversity into system for use as cyber maneuvers in a moving target defense, which employs usage of control theoretic principles. This work discusses maneuvers including memory randomization, IP address randomization and application of a new state machine with random extra states for protocols such as DHCP. In [14], a concept of Random Route Mutation (RRM) is introduced and algorithms are defined to achieve optimal path randomization between a source and a destination. In [94], a basic design schema of a moving-target network defense system is presented and a simulation-based study is conducted to explore the degree to which proactively changing a network's various parameters can decrease an adversary's chance for success.

2.2 Distributed and Parallel Computing Technologies

In this section, we present an overview of the distributed and parallel computing technologies and their associated challenges. This background provides motivation for our research efforts. We cover software-defined networking, the MapReduce programming paradigm, and scheduling systems for cluster and shared computing resources.

2.2.1 Software-Defined Networking

Software-defined networking (SDN) technologies such as OpenFlow [60] have emerged in the last few years as a promising new approach to operating computer networks. The seminal feature of SDN is centralized control of packet forwarding via software controllers. Given centralized knowledge of the complete network topology and traffic demand, optimizing and fault proofing of traffic forwarding can be conveniently implemented and invoked in the SDN controllers. Unlike creation of a new protocol or protocol stack, introduction of SDN is a change of paradigm – as its centrally optimized and proactive control methodology presents a stark difference to today's distributed and reactive Internet architecture. A centralized network controller is able to programmatically modify a switch or routers handling of traffic based on frame, packet or flow. Coupled with high speed networking technologies (100 Gbps in the Internet core and 10 Gbps to the edge), the centralized control methodology makes an even stronger case for achieving extremely high end-to-end networking performance via central optimization.

Pioneering SDN research efforts explore use cases for network virtualization [82], service insertion [6, 56, 77], load provisioning [44, 90], and network debugging [43]. Our work in Chapter 4 uses SDN technologies to maneuver the physical and logical topology of a computing cluster for application optimization.

In the context of data-intensive computing, the most direct benefit brought by SDN is the ability to flexibly and reliably control network bandwidth amongst compute hosts and storage [46]. The collection of the last two years' research presented at top conferences for cluster, grid, HPC and cloud computing have arrived at the following common conclusions: first, virtual machines running in modern data centers are increasingly the desired vehicle for data intensive computing due to their clean OS isolation, fast configuration, pause, migration, termination, and low setup overhead compared to direct execution over bare metal servers [36]. Secondly, benchmark tests have repeatedly confirmed performance bottlenecks due to low disk I/O or network throughput instead of processing capacity [46]. Finally, enabling virtual machines with dedicated resources, including processor cores, storage, and network bandwidth significantly enhances performance [19].

The research trend clearly suggests that effective resource allocation in a data-intensive distributed environment is a top priority for the community. While allocating processors for data-intensive computing tasks using batched task schedulers is a well-learned practice, the same is not as well understood for storage and network resources. To date, the majority of data centers provide best-effort network throughput among processors and aggressively over-provision the raw network bandwidth to achieve acceptable performance for users. Where high network throughput is absolutely needed, it is offered as a premium service by placing the tasks on separate carefully admission-controlled hardware with high-speed connectivity. This practice results in a substantial loss of potential computing capacity and, even more crucially, severely limits HPC data centers' ability to host time-sensitive applications. In the case of scientific workflows in such disciplines as genomics, synthetic chemistry, atomic physics and beyond, very large data transfer is the cause for severe bottlenecks. Today, this is addressed by custom planning and over-provisioning of networks among dedicated HPC centers for the exact data flows expected. The approach does not scale beyond a few heavily invested sites, such as those established for the LHC project [24]. To overcome such limitations, methodologies for joint allocation of compute, network, and storage resources must be studied.

2.2.2 Google MapReduce

MapReduce is a programming model for implementation for computing across large data sets in a parallel and distributed manner. It is built upon the functional programming methods of *map* and *reduce*. MapReduce is attractive to maneuverabilities studies due to the level of autonomy and redundancy of the individual operations.

In 2004, Google presented their seminal work on the MapReduce Programming Model [28]. MapReduce is the parallel programming paradigm used to build the index of the World Wide Web, which is a cornerstone in Google search. This paper provides the insight for multiple MapReduce software suites to be built by the community.

A MapReduce job is made up of multiple map tasks and multiple reduce tasks. A mapper takes an input set of data in a key-value pair (K, V) and outputs an intermediate key-value pair

(K', V'). The input to the reducer is an intermediate key and the set of intermediate values for that key $(K', \{V'_1, V'_2, V'_3, ...\}$ from all the mappers across the cluster. The reducer performs some computation on the set of intermediate values and produces the final output value for each key for which it was responsible for reducing. The entire possible set of intermediate key values are partitioned and each partition is assigned to a reducer. During the shuffle phase between map and reduce, a reducer pulls its respective partition from each mapper before beginning the reduce computation.

2.2.3 Apache Hadoop

Hadoop MapReduce [37], the de-facto standard infrastructure support tool for MapReduce, is a network-based parallel programming paradigm for implementing data computation over big data using a cluster of commodity computer systems. Many industrial and governmental organizations have built large-scale production data centers with dedicated computing resources for Hadoop clusters to support their big data and scientific computation workloads.

The network file system for the cluster is the Hadoop Distributed File System (HDFS). This file system is composed of a single centralized management node, *NameNode*, which maintains all the metadata for the cluster along with multiple storage nodes, *DataNodes*, which contain all of the data in large block sizes (default 64 MB). Large files are divided into blocks and the data blocks are replicated across the cluster to provide redundancy.

The MapReduce computation model includes a single central management node, *JobTracker*, and multiple computation nodes, *TaskTrackers*. The JobTracker is responsible for delegating the specific map and reduce task for a submitted MapReduce job to a subset of the TaskTrackers in the cluster. The JobTracker further monitors the status of the TaskTrackers to ensure redundancy and timely completion of the job. A single TaskTracker can be assigned both map and reduce tasks within the same job. DataNodes and TaskTrackers exist on the same physical computing system, thus providing the computation and storage integration that is critical to the performance of MapReduce [37].

The JobTracker for a MapReduce cluster, in concert with the NameNode, is data locality aware for assigning map tasks, but by default does not consider data locality for assigning reduce tasks. Data locality awareness means the JobTracker assigns map tasks to the TaskTracker, which is home to the input data in HDFS. This allows the computation to be executed on top of the data with no network transfer. As such, TaskTrackers can begin executing as soon as they are assigned a map task. A reducing TaskTracker must wait until all mapping TaskTrackers have finished before retrieving their input partition. Thus total execution time is affected by the last reducer to complete. TaskTrackers are assigned reduce tasks and specific partition on a first come first serve basis, with no consideration to map output data location.

The division of possible intermediate key values is customizable by the user on a job-by-job basis in a MapReduce cluster. This functionality is provided by a Partitioner. The default Partitioner for MapReduce is the HashPartitioner. The HashPartitioner applies a hash function to the key and then computes the remainder when the number of reducers divides the hash value. This modulus math function provides the index of the R reducer that is responsible for reducing the set of values associated with this key. This partitioner expects a uniform distribution of keys across the set of reducers, in theory assigning each reducer $\frac{1}{R}$ of the intermediate keys. This partitioner does not consider the size of the intermediate values that is associated with each intermediate key. The difference between the actual measure of the partition size and the theoretical size for each reducer is called the reducer partition skew. This skew can be due to the nature of the input values, the intermediate key set and associated size of the intermediate values.

Hadoop is currently in version 2, which includes enhancements to HDFS and introduces additional programming models beyond MapReduce [88]. The implementation of MapReduce in version 2 remains consistent with the features described. Though our work in Chapter 3, 4, and 6 focuses on Hadoop v1, it can be applied to the MapReduce features of the latest version.

2.2.4 Improving Shuffle Bottleneck

In a MapReduce job, map tasks are assigned to nodes such that the input data is stored locally. Output from map tasks must be moved across the data center network in order to reach task nodes that preform the reduce tasks. This bulk data transfer places significant burden on data center network and can be a major component to task completion time. The phase where output from map tasks is transferred to reduce task is called the shuffle phase.

Two recent publications have introduced creative approaches to addressing the network bottleneck during the shuffle phase of MapReduce jobs. The first approach integrates software defined networking in the data center to establish fast optical circuit switched paths between racks of mapper and racks of reducers [89]. This reconfiguration of the physical and logical paths promises to provide large communications channels for shuffle phase bulk data transfers. The second approach modifies the MapReduce job's center control node to schedule reduce tasks as close to mappers that contribute a significant portion of the reduce task input data. [40, 41]

Wang, Ng, and Shaikh [89] have proposed addressing the shuffle phase network bottleneck by using a combination of software defined networking in a hybrid electrical optical networked data center. The central management structure of HDFS, MapReduce, and software defined networks promises an integration strategy to allow the network to be adapted at run-time for optimization of big data applications. Though no actual implementation is described in the paper, they provide theoretical results, which suggest such an approach could greatly benefit shuffle times with this system. The results in [89] help to motivate our proposed research as described in Chapter 4.

Hammoud, Rehman, and Sakr [40, 41] suggest a different approach to limiting the impact of data center bandwidth bottlenecks on MapReduce job performance. They address the reduce task scheduling of a Hadoop cluster to provide data locality and data skew awareness. In native Hadoop, data locality is afforded to map task scheduling, but not reduce task scheduling. Their system, the Center-of-Gravity Reduce Scheduler (CoGRS), has been implemented and has shown to reduce off-rack network traffic in their private cloud by 9.6%.

Additional research into improving Hadoop shuffle traffic has focused on the partitioning function [49]. Many works have looked at OpenFlow enhancements for Hadoop traffic in general [57, 68, 91]. Above and beyond Hadoop, there is increasing interest in the topology of data centers with researchers focusing on decreasing cost, improving application performance, and lowering power consumption. New data center topologies have been proposed [12, 13, 39, 66] while others have proposed SDN enhancements to the flows within current topologies [92]. Finally, research is even further growing in understanding network characteristics and optimization in production data centers [18, 20]

2.2.5 Portable Batch System

Many scheduling algorithms and systems exist to allow computing resources to be shared between multiple user groups. Our work in Chapter 3 utilizes the Palmetto Supercomputing Cluster at Clemson University. Our cluster uses the Portable Batch System for scheduling jobs. We provide an overview of the PBS in this section.

The Portable Batch System, PBS, was developed by the Numerical Aerodynamic Simulation

Facility at NASA Ames Research Center in the 1990s [45]. PBS provides an external batch scheduler for execution of shared supercomputing resources. PBS maintains different priority job queues with designed authorized users and hardware. This allows administrators to give some jobs to be given preferential scheduling over other jobs and to limit the runtime of user submitted jobs. Users submitting jobs to PBS are allowed to specify the number of computing nodes, the number of processors, and memory needed on each computing resource, among other options.

PBS jobs can be interactive or can run in batch mode, executing designed programs or scripts that are provided as part of the job submission. Typical parallel programming models involve the creation of a set of parallel processes, a designated set of data, and the location for storing the output. Jobs are submitted to a queue. When serviced the job is assigned to a set of nodes to be executed. The user does not have to concern himself or herself with the job schedule. That is the domain of the scheduler. While a job is running, there is the potential for the job to be preempted by a job submitted to a queue with a higher priority that needs the preempted job‘s computing resource.

2.2.6 Deploying Hadoop in Shared Resources

Our work in Chapter 3 applies maneuver to provision a long-running Hadoop cluster within the administrative and financial constraints of an existing high performance computing environment. Below we review some current approaches to deploy Hadoop within cloud, HPC, and grid systems.

A common solution to integrating Hadoop into existing HPC clusters is the use of virtualization. For clusters that support virtualization interfaces, users can reserve a set of compute nodes and set up a cluster of virtual machines (VMs) on these nodes. This provides users with an isolated environment in which they have full root-level control over the set up of Hadoop. Notable HPC clusters that take advantage of this approach are the FutureGrid distributed testbed [30] and the Amazon Elastic MapReduce Cloud [1]. A drawback of this approach is the degradation in I/O performance for data-intensive MapReduce applications, which has been observed on FutureGrid [54]. Another potential issue is the interaction between Hadoop and non-Hadoop jobs that are part of a workflow using non-Hadoop software packages installed on the physical hardware.

Another approach is to dynamically set up Hadoop environments in users' workspace. A solution provided by Apache is the Apache Hadoop on Demand (HOD) system [4]. HOD enables the quick provision and management of multiple Hadoop MapReduce instances on a shared cluster with PBS Torque scheduler. However, HOD requires access to a static external HDFS cluster. The myHadoop system extends HOD by including a dynamic HDFS with user-generated Hadoop environments [55]. This HDFS could either be placed on the local storage of the compute nodes in non-persistent mode or on a permanent external parallel file system in persistent mode. The approach of HOD and myHadoop enables the concurrent and isolated creation of Hadoop environments with dedicated resources through reservations. However, the implementations of HOD and myHadoop do not allow users to automatically deal with administrative and policy issues such as walltime limitation and priority preemption of computing resources.

Recent work focuses on creation of a new resource management mechanism that can handle both Hadoop and non-Hadoop parallel jobs. Significant efforts include the Apache Mesos project [47] and Apache Hadoop YARN (Yet Another Resource Negotiator) [38]. The main principle for this approach is the separation of resource management and task scheduling mechanism in Hadoop's JobTracker. The new stand-alone resource manager can be used to handle both Hadoop and non-Hadoop processes. This keeps the users from having to configure a new Hadoop environment every time while still maintaining dynamic and isolated execution of MapReduce applications [3].

2.3 System Modeling Tools

There are many modeling techniques available to researchers including but not limited to Markov Chain analysis and finite state automata. One very popular model that is equivalent or includes most other models is the Petri Net. Petri Nets are uniquely suited for modeling system with concurrent and asynchronous actions, such as distributed and parallel systems and applications [74]. Petri Nets provide a simple graphical representation that makes not only analysis but also design of systems convenient. There are multiple automated tools and solvers available to analyze Petri Nets [31]. Additionally, many cybersecurity systems have been modeled with Petri Nets in the literature.

2.3.1 Petri Nets

Petri Nets were created by Carl Adam Petri [67] as a method to study concurrency in parallel and distributed applications. Petri Nets have been applied in many disciplines and extended in multiple facets through the years. Petri Nets are found in use within the computer science field to model of a distributed computing system in which to study the correctness, concurrency, and synchronization.

A Petri Net is a bipartite, weighted, directed graph composed of two types of nodes called Places and Transitions and edges called Arcs. Places represent preconditions or values of variables in the system and are designated as circles graphically. Transitions represent actions taken as a result of valid preconditions or values and are represented as rectangles or lines. Since the graph is bi-partite, arcs only exist between places and transitions. There are no arcs between two places or two transitions. Places from which an arc originates are considered input places to the transitions in which the arc terminates. Places in which arcs terminate are considered output places of the transition at the origin of the arc. Tokens, represented by solid dots, indicate preconditions being true or values of variables. Multiple Tokens can be found in places throughout a net. The distribution of tokens over a net represents the configuration of the system and is called the marking. Arcs have weights that represent the number of tokens consumed or produces as a result of transitions "firing". When a transition "fires" tokens from its input places are consumed and tokens are produced in the output places. Transition can only fire when enabled, meaning all input places have the minimum number of required tokens. Transition firings are atomic and nondeterministic. Multiple transitions could be enabled concurrently, but the order in which the fire is not known and the result of a transition firing can result enable or disable other transitions.

A basic Petri Net with four places and two transitions is show Figure 2.1. In the current marking of this Petri Net, $P0$ has two tokens, $P1$ and $P2$ each have one token, and $P3$ has no tokens. The weight of two on the arc from $P0$ to $T0$ indicates that two tokens in $P0$ must be present for $T0$ to be enabled to fire. The weight of 2 on the arc from $T1$ to $P3$ indicates that if $T1$ fires that it will place two additional tokens in $P3$. These tokens are added to any tokens that may already be present in $P3$.

A marking of a Petri Net is indicated as M which is a vector of length m which is the number of places in Petri Net. Each value of m_i represents the number of token present in the ith place. Each Petri Net has an initial marking, M_0, which describes the initial state of the system. A marking, M', is said to be reachable if a series of valid transition firings from M_0 results in M'. A way to visualize all reachable states is to create a reachability graph that shows all reachable markings and the transition firing sequence to each state. This graph shows the entire state space for a Petri Net. The reachability graph for the basic Petri Net presented above is shown in Figure

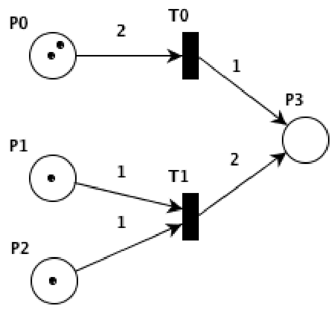


Figure 2.1: A basic Petri Net with four places, two transitions and four arcs.

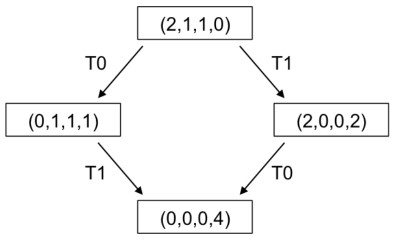


Figure 2.2: Reachability Graph for basic Petri Net in Figure 2.1.

2.2.

2.3.2 Stochastic Petri Nets

Multiple extensions of the basic Petri Net can be found in throughout the literature [52,75]. One such extension, is the Stochastic Petri Net (SPN) [11] where transitions have a delay between enabling and firing. This delay is recalculated each time the transition becomes enabled and has an exponential distribution where each transition is assigned its own firing rate. When multiple transitions are enabled, the transition with the shortest delay time will fire first. Graphically, the timed transitions are shown as unfilled rectangles, as opposed to solid rectangles or straight lines which are immediate transitions in the standard Petri Net. Figure 2.3 shows a sample SPN with firing rates $(\lambda_0, \lambda_1, \lambda_2)$.

An SPN in which the number of tokens in a place is finite has been shown to be equivalent to a finite Markov Chain. The Markov Chain can be determined from the reachability graph of the SPN and the initial marking. This makes it possible to solve the steady-state distribution of the SPN. This steady-state distribution allows one to analyze systems described as stochastic Petri Nets to determine probability of the system existing in a given state. Table 2.1 formally describes

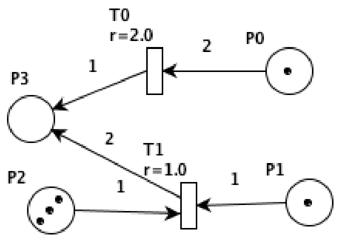


Figure 2.3: A Stochastic Petri Net with firing rates for each transition shown

a Stochastic Petri Net as a 6-tuple PN.

Table 2.1: Formal Definition of Stochastic Petri Net

$SPN = (P, T, R, I, W, M_0)$
$P = \{p_0, p_1, ..., p_m\}$ is a finite set of places,
$T = \{t_0, t_1, ..., t_n\}$ is a finite set of transitions,
$P \cap T = \emptyset$,
$\lambda = \{\lambda_0, \lambda_1, ...\lambda_n\}$ is the set of transitions firing rates,
$I = I^- \cup I^+$
$I^- \subseteq (P \times T)$ and $I^+ \subseteq (T \times P)$ are a sets of input and output arcs,
$W : I \rightarrow \{1, 2, 3, ...\}$ is a weight function
$M_o : P \rightarrow \{0, 1, 2, 3, ...\}$ is the initial marking.

2.3.3 Platform Independent Petri Net Editor (PIPE2)

The Platform Independent Petri Net Editor (PIPE2) [31] began as a graduate course group project at the University of London in 2003. The goal of the project was to design an application that allows Petri Nets to be designed graphically and analyzed.

PIPE2 is written in Java and compatible with any OS capable of running Java programs. The system provides the capability to create standard Petri Nets and Stochastic Petri Net models. The tool allows a user to draw a SPN using drag-and-drop tools on a canvas, then to save the file in an XML file that can be opened in other applications. The tool also has the ability to animate the model with random firing of transitions or interactive user manipulations. The key aspect of the tool is the wide variety of analysis modules and the ability for users to design and integrate their own custom modules. One such module is the Steady State Analysis. This module determines reachability graph of a SPN then calculates the steady state distribution. We use this module in our analysis in Chapter 5.

2.3.4 Petri Nets for Security Modeling

Some research efforts have been devoted to security modeling and analysis using SPN [25, 27, 86]. In [27], an approach using generalized stochastic Petri nets (GSPNs) to model and analyze attack trees is proposed with the ultimate goal of automating the analysis using simulation tools. The results of this simulation and analysis can then be used to further refine the attack tree or to develop corresponding countermeasures. In [86], a vulnerability assessment framework is proposed to systematically evaluate the vulnerabilities of SCADA systems at three levels: system, scenarios, and access points using a generalized stochastic Petri net model. The proposed method is based on cyber systems embedded with the firewall and password models, the primary mode of protection in the power industry. In [25], the authors investigated the use of Petri nets for modeling coordinated cyber-physical attacks on the smart grid. A hierarchical method is provided to construct large Petri nets from a number of smaller Petri nets that can be created separately by different domain experts for analyzing cyber-physical attacks.

2.3.5 Other Maneuverable Projects

Research investigating applications having maneuvering characteristics have been funded by the U.S. government and can be found in the literature. These works concentrate on single node applications, where our work concentrates on parallel and distributed applications. Additionally, these works focus on providing secure environments only, where we extend the idea of maneuverability into the realms of resource provisioning and application optimization.

Raytheon's Net Maneuver Commander (NMC) [21, 22] is a prototype command and control system enhanced by moving target defense. NMC inserts randomness into hardware platforms, operating systems, and network segments to move network-based elements increasing resiliency. The system has been used to maneuver military command and control systems, databases, and network services such as DNS, DHCP, web proxies, and VOIP servers. Additionally, they provide a honey net implementation that moves previously used nodes into a monitored and controlled network segment for isolation and identification of potential attackers.

Their work presents an overview of their motivations and architecture. They present analytic results of simulated force-on-force interactions. By introducing an analysis framework [42] and metrics to provide feedback and measurement of the improved resiliency of the environment, they show the NMC does lower the percentage of successful attacks and increase the time to complete successfully attacks. Finally, they present recommendations on how to expand the work and deploying the system in production environments.

Lincoln Laboratory's Trusted Dynamic Logical Heterogeneity System (TALENT) [71] is a framework for live migration of applications across a set of heterogenous systems. TALENT by utilizing systems with different instruction sets and operating systems increases the dynamic attack surface of the critical application. TALENT implements portable checkpoints and operating system level virtualization migration to achieve maneuver of critical systems, prioritizes mission success over individual system stability. By preserving internal state of an application, TALENT can migrate entire environments including but not limited to virtual networking, open file handles, transferring buffers, and signals. TALENT does not contain threat detection mechanism to trigger migrations, only randomized movements.

Both, NMC and TALENT, differ from our proposed work with the Defensive Maneuver Cyber Platform in that we are focused on a distributed and parallel application where individual nodes maneuver between active, idle or deceptive membership. We also are explicitly modeling a system in search of analytical findings of increased survivability in the presence of attackers. Additionally, our work utilizes deception as a feedback mechanism to understanding the threat conditions and adapting the system at run time relative to the conditions.

Our work also differed from NMC and TALENT by using maneuver for resource provisioning and application optimizing. Many research efforts have focused on relating cyber maneuver to only moving target defense and cybersecurity efforts, where we have shown maneuver can be employed for additional system advantages.

2.3.6 Summary

This chapter presents a survey of background information in use by our completed work and our proposed work. Specifically, we review military cyberspace issues and research, distributed and parallel system technologies, and system modeling tools.

Chapter 3

Maneuver for Resource Provisioning

The first area of interest for studying maneuver in cyberspace platforms is in the area of resource provisioning. Our work with the Job Uninterrupted Maneuverable MapReduce Platform shows how we can provision a big data environment in a university setting within the current investment of high performance computing. This is achieved by the use of maneuvering of nodes in and out of the cluster.

JUMMP, the Job Uninterrupted Maneuverable MapReduce Platform [65] is an automated scheduling platform that provides a customized Hadoop system within a batch-scheduled cluster environment. JUMMP enables an interactive pseudo-persistent MapReduce platform within the existing administrative structure of an academic high performance computing center by "jumping" between nodes with minimal administrative effort. Jumping is implemented by the synchronization of stopping and starting daemon processes on different nodes in the cluster. Our experimental evaluation shows that JUMMP can be as efficient as a persistent Hadoop cluster on dedicated computing resources, depending on the jump time. Additionally, we show that the cluster remains stable, with good performance, in the presence of jumps that occur as frequently as the average length of reduce tasks of the currently executing MapReduce job. JUMMP provides an attractive solution to academic institutions that desire to integrate Hadoop into their current computing environment within their financial, technical, and administrative constraints.

3.1 Introduction

This rapid evolution and adoption of Hadoop has created a rich and complex software ecosystem [61]. This introduces challenges to system administrators in offering this service while maintaining a stable production environment. This is especially challenging in a typical centralized academic research environment where the hardware and software infrastructures are designed to accommodate a wide variety of research applications within a set of financial, technical, and administrative constraints. To address this problem, we introduce the Job Uninterrupted Maneuverable MapReduce Platform. JUMMP is an automated scheduling platform that enables the integration of Hadoop into the existing large scale computational infrastructure to support high availability and continuous computing for research and education.

The strength of Hadoop comes from its design characteristics that bring computation to data stored on large local hard drives to take advantage of data locality instead of having to transfer data across the network, and to guarantee job completion in the event of frequent failures. This comes at a cost of extra computing cycles due to additional communication and metadata overhead to support very large scale transparent parallelization [73]. As a result, a typical Hadoop cluster consists of computing nodes with multiple terabyte-sized local storage [48]. Standard practice for architectural design of centralized, shared academic research environments usually places the computing components and the storage components into separate clusters. The local storage on the computing components is temporary and measured in just a few hundreds of gigabytes. Provisioning Hadoop as a separate stand-alone cluster requires the additional acquisition of new hardware, new rack placements, and additional power and cooling cost. On the other hand, the setup of a dynamic Hadoop environment is limited to standard resource scheduling policies in a shared computing environment. Examples of such policies are the maximum number of resources that can be reserved, or the maximum amount of time that a computing node can be reserved. This places a limit on the capability of Hadoop-based programs within such an environment.

Our work with the Job Uninterrupted Maneuverable MapReduce Platform provides the following contributions:

- A platform that enables the integration of Hadoop into an existing large scale academic computational infrastructure,
- Automated scheduling mechanisms that facilitate an arbitrarily long run time of MapReduce

applications within the administrative constraints of a shared environment and with minimal interactions required from system administrators, and

- Experimental results that demonstrate that jumps can occur as frequently as the average length of the reduce tasks of the executing MapReduce jobs, with only modest impact to performance.

3.2 Motivation

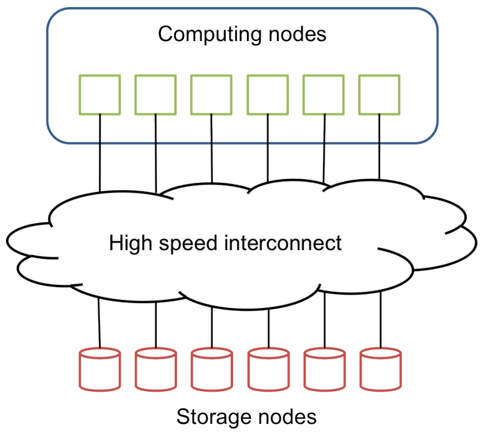
In this section, we describe user and environmental considerations that motivate the JUMMP implementation. These considerations are observed in the typical shared high performance computing environment found in academic institutions.

3.2.1 User Considerations

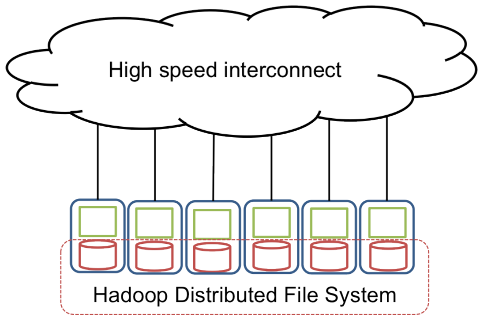
In an academic environment, we can classify the usage of Hadoop into three different categories. The first category includes research applications that use Hadoop MapReduce as a tool for research purposes. These projects can either use MapReduce programs exclusively or use MapReduce programs as part of a larger workflow in a programming framework. The researchers may spend some time developing MapReduce programs and other necessary components and then focus on executing the programs to achieve the final results. As these are research applications, researchers are typically alternating between running the computations and analyzing the produced outputs.

The second category is the study of the Hadoop MapReduce ecosystem itself. This includes studies of Hadoop MapReduce under different hardware and software configurations, development of improvements to Hadoop MapReduce, and implementations of different alternative parallel frameworks to Hadoop MapReduce. While these projects do not usually use as many computing hours as the first category, they need to frequently test different setups of the Hadoop cluster for performance analysis purposes. This testing of dynamic execution environments for Hadoop is difficult to do in most dedicated production facilities.

The third category includes small projects such as those submitted by students doing course projects. While these projects usually have short run times and use small data, the nature of the students' behavior can lead to unintended issues. In our experiences teaching Hadoop MapReduce during the Fall 2012 and Spring 2013 semesters, we observed multiple problems, such as crashing of



(a) A typical HPC cluster with compute nodes separated from storage nodes



(b) A Hadoop cluster with storage on the compute nodes for data locality

Figure 3.1: Comparing a design choice for computing and storage placement between typical HPC cluster and Hadoop cluster

the Hadoop MapReduce core processes, corruption of data on the cluster's HDFS, and overloading of the cluster as students rush to complete the work before deadlines.

With these categories, we have the following design objectives:

- Support guaranteed execution of MapReduce programs until completion regardless of run time (currently, there is no check-pointing mechanism for MapReduce jobs),
- Allow reconfiguration of Hadoop environment settings such as block size, number of map tasks, and number of reduce tasks, for different performance evaluation experiments,
- Facilitate isolated execution of MapReduce programs, and
- Support rapid movement to different hardware when impending failures are detected on current hardware.

3.2.2 Environment Considerations

While considering the needs of the users, we also seek to address other constraints of a typical academic high performance computing environment. The first constraint is cost. The costs of purchasing new computing hardware, building new floor space, and maintaining power make it difficult for many academic computing centers to provision dedicated resources for Hadoop.

A second constraint is the difference in technology between Hadoop MapReduce and a typical high performance computing cluster. Standard accepted practice for building high performance computing clusters (HPC) is to separate compute nodes from storage nodes. The compute nodes handle CPU-intensive parts of parallel applications while the storage nodes handle I/O demands of the applications, typically through a parallel file system [81]. Parallelization mechanisms are facilitated by specialized libraries at run time (e.g. MPI libraries for C-based applications), and data movement between computation and storage is done through a high-speed interconnect. While Hadoop MapReduce also relies on specialized libraries to facilitate parallelism, its performance comes from the data locality enabled by the Hadoop Distributed File System (HDFS). HDFS divides data files into equally sized blocks and stores them on the local hard disks of compute nodes. Instead of transferring data through the network, the computation processes are spawned locally and assigned to individual blocks. This difference in computation and storage placement is illustrated in Figure 3.1.

A third constraint is the support of the run time environment of Hadoop. Hadoop maintains two permanent Java daemon processes, called the DataNode and the Task Tracker. These process are on all of the compute nodes at all times in order to handle both HDFS and MapReduce. The continuous execution of dedicated persistent Java virtual machines makes Hadoop unsuited for a centralized installation on an HPC cluster where users also frequently run non-Hadoop applications.

Our approach is to provision Hadoop as a dynamic execution environment that can be set up, executed, and shut down on demand by a user. This is feasible from user space since Hadoop's core Java processes do not require any root-level privileges to execute and communicate. However, scheduling dynamic setups of Hadoop environments creates overhead due to run-time configuration, data staging, output write, and shutdown of the environment. Related work shows that for data-intensive applications, the total overhead could be as high as the execution time itself [55]. To make dynamic provisioning effective, there must be a mechanism that allows dynamic Hadoop environments to reserve the resources beyond the standard policy of per-user limits in a semi-persistent manner during the computation phases of application execution. An obvious solution to this problem, permanently assigning a set of compute nodes to specific users, places additional responsibilities on the administrators such as how to set a scheduling policy for users who want to execute both Hadoop and non-Hadoop applications, and how to make sure that the allocations of computing resources are fair.

Our goal is to facilitate the setup of user-controlled dynamic Hadoop environments that execute within existing scheduling policies including resource limitation, maximum walltime, and priority preemption, without administrative intervention.

3.3 Architecture

3.3.1 JUMMP Design

JUMMP is designed to operate within the Palmetto High Performance Computing Cluster at Clemson University [7] using the PBS Professional scheduler, although the system can easily be adapted to use with other schedulers. Hadoop version 1.1.2 is used in our system. Any 1.x version can be used. The Hadoop cluster uses a single dedicated node for both the NameNode and JobTracker. This dedicated node resides outside the control of the scheduler. Each DataNode/TaskTracker is scheduled as an individual PBS job, which allows for preemption or failure of a PBS job to only affect a single node of the Hadoop Cluster.

The design choice to use a dedicated persistent node as the master of the Hadoop cluster is based on an assumption that securing a single node for dedicated use from an HPC administrator is more achievable than securing all the nodes required for the cluster. Also, the transition of the head node of an active Hadoop cluster during the middle of a MapReduce job is technically more difficult than the transition of the slave nodes. Including the JobTracker and name node in the maneuvering portion of the cluster is a topic for future work.

Each DataNode and TaskTracker is established within its own PBS job. The PBS job starts the Hadoop daemons and connects them to the persistent head node. Each daemon waits for its trigger to jump. When the trigger to jump is received (PBS preemption or expiration of the jump timer), the PBS job schedules its replacement PBS job. Next, the DataNode decommissions itself from the name node and the TaskTracker blacklists itself from the JobTracker. Finally, the Hadoop daemons are stopped and the PBS job completes. The newly spawned replacement node repeats this process.

When notified of an upcoming jump of a slave node, the name node begins to reassign the blocks replicated on the outgoing node to the remaining nodes in the cluster. The incoming node receives a subset of the blocks its previous node possessed. The JobTracker immediately kills any task assigned to the outgoing TaskTracker and reassigns those tasks to available TaskTrackers.

In our earlier implementations of JUMMP, we stopped the TaskTracker daemons without notifying the JobTracker. After the standard heartbeat failure timeout, tasks would then be re-assigned. This behavior greatly degraded performance. The proactive measure of blacklisting the TaskTracker allows the JUMMP to transition much faster.

JUMMP is implemented using a combination of Bash and Python scripts to control execution of the daemons, logging of actions, and trapping of signals to ensure the rescheduling of replacement nodes during jumps.

We describe the configuration of the JUMMP with the following variables:

- The number of DataNodes and TaskTrackers in the system is n.
- The scheduled time between node jumps is t_j.

When the JUMMP system is initialized, the user is able to specify n and t_j. After a stabilization period, a single node lives for nt_j minutes. A node jumps somewhere in the cluster every t_j minutes.

In the event of a jump caused by an PBS scheduler preemption, the jumping node could alert its new node of the remaining time until the regularly scheduled jump, thus allowing the timing of the cluster transitions to be resumed. This topic is an area of future work.

The flexible and maneuverable design of JUMMP allows us to support our user considerations as presented above. While allowing preempted or failed nodes to submit PBS jobs for their replacement, we have guaranteed survival of the Hadoop cluster and continuation of running jobs until completion. Since each instantiation of JUMMP specifies a Hadoop directory with executables and configurations, users are able to tailor their JUMMP instance to the needs of their long running Hadoop environment on an isolated set of nodes within the shared cluster environment. Lastly, administrators can "blacklist" nodes that are experiencing prolonged failure or which have upcoming maintenance. Blacklisted nodes are not listed as available resources for the scheduler, thus allowing JUMMP to avoid and maneuver around those resources to more suitable hardware.

3.4 Experiment

Adding maneuverability to a Hadoop cluster incurs some degradation of performance. We attempt to evaluate this degradation to understand the trade-offs of such a system and to estimate the rate at which a jump could efficiently occur. With each jump of a DataNode, additional overhead is occurred over a non-jumping cluster. This overhead comes from two separate sources: the scheduler overhead and replication of HDFS blocks. As previously mentioned, JUMMP schedules each DataNode and TaskTracker as separately scheduled jobs within the supercomputing environment. Scheduling and queuing delays results in fewer TaskTrackers being available to perform map and reduce tasks. We refer to the JUMMP as being "undersized" during the time when a node is awaiting the scheduler to place the slave node on an available node in the datacenter. Once an existing DataNode leaves the cluster, all blocks that were stored on its local storage must be replicated across the cluster. The NameNode begins this immediately upon the decommissioning of the outgoing node. This data replication consumes processing, network bandwidth, and disk drive seeks that normally would be used for computation for the currently running MapReduce job.

3.4.1 Design

In order to capture, measure and quantify this degradation, we conducted two separate experiments on the performance of JUMMP on the Palmetto High Performance Computer Cluster. In each experiment, we establish a baseline performance metric for a stationary Hadoop cluster by repeatedly running the same MapReduce job 100 times over a static dataset. We rerun the job three additional times while varying the jump time for the cluster. We record the times of the jumps, the individual task start and stop times, and the overall job run time. With these results we quantify the overhead of jumping during the execution of a MapReduce job.

We ran our experiments on a homogeneous set of nodes within Palmetto to ensure uniformity of test results. All tests are run on an isolated pool of 96 nodes for DataNodes and TaskTrackers. The hardware of these nodes is shown in Table 3.1. A datablock size of 256 MB is used throughout the experiments and each TaskTracker has 8 map task slots and 4 reduce task slots. 2GB of memory is allocated for each task. At the allocation of the initial nodes and creation of the JUMMP, the dataset for the experiment is imported, the nodes start jumping, and the MapReduce job begins to run.

For our experiments, we use the dataset and benchmarks from PUMA, Purdue's MapReduce Benchmark Suite [10]. PUMA is developed as a benchmark suite to represent a broad range of MapReduce applications exhibiting application characteristics with high/low computation and high/low shuffle volumes. The parameters of our two experiment runs are shown in Table 3.2. The

Node	HP SL250s
CPU	Intel Xeon E5-2665 (2)
Cores	16
Memory	64 GB
Local Storage Capacity	900 GB
Networking	Infiniband

Table 3.1: Node Configuration

Application	Wordcount	Terasort
Dataset Size	50 GB	300 GB
Node Count	8	32
Jump Times [mins]	7/10/15	20/40/60

Table 3.2: Experiment Parameters

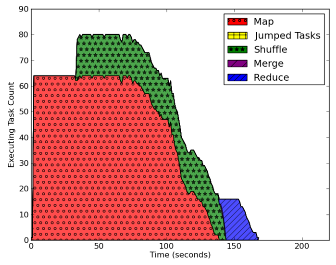
jump times are calculated based on the average running time of the baseline MapReduce job. Jump times are configured to jump during every second, third, and fourth job runs.

3.5 Analysis

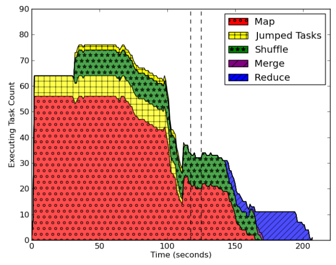
3.5.1 Characteristics of the Impact

Whenever an active TaskTracker jumps, all tasks currently executing and some previously completed tasks have to be rerun. Since all reduce tasks must retrieve their partition of intermediate data from all map tasks, any map task previously completed by a jumping TaskTracker have to be rerun if all reduce tasks have not finished the shuffling phase. Furthermore, any reduce tasks being executed by the jumping TaskTracker have to be reassigned and rerun. If any reduce task is restarted, then then the map task whose intermediate data is unavailable also have to be rerun. This re-execution of in-progress and completed tasks adds delay and overhead into the overall execution of the MapReduce job.

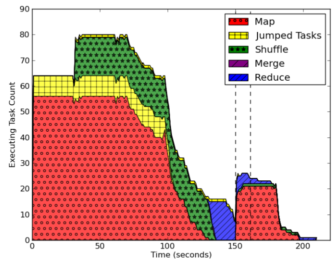
This restarting overhead is visualized in Figure 3.2. Figures (a) and (d) show a typical execution of the *wordcount* and *terasort* experiments in the absence of a node jump. The charts plot the number of currently executing tasks, grouped by type, during the time of the job execution. The job starts with multiple map tasks conducted in parallel. Once the first set of map tasks completes, the shuffle phase of the first set of reduce tasks begins downloading their partitions of the intermediate data from the completed map tasks. This process continues until all map tasks have completed and the shuffling is done, at which time the reduce actions begin. The reducing continues until all reduce tasks have completed and the output is written to HDFS.



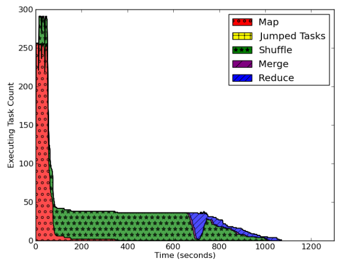
(a) No jump during wordcount execution



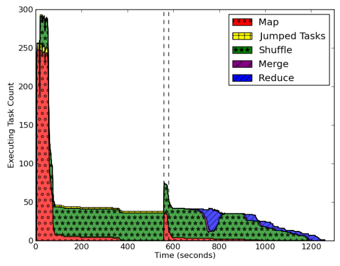
(b) Jump during map phase of wordcount execution



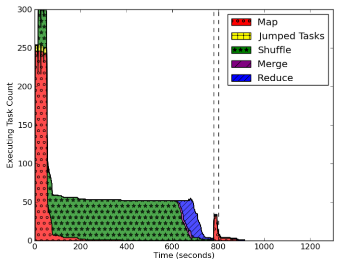
(c) Jump during reduce phase of wordcount execution



(d) No jump during terasort execution



(e) Jump during map phase of terasort execution



(f) Jump during reduce phase of terasort execution

Figure 3.2: MapReduce job task execution during jump

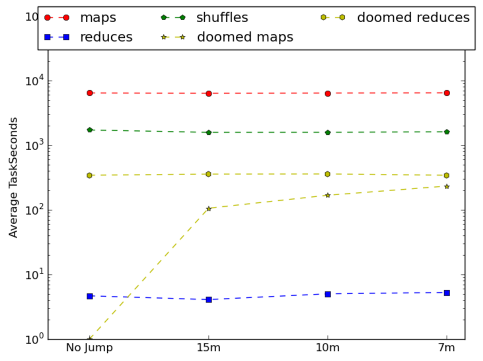
Figure 3.2 (b) and (e) present the cost of jumps occurring before all the mapping and shuffling phases have completed. In these examples, a spike in the number of map tasks at the time of the jump (indicated by dashed vertical lines) shows that additional mapping and shuffling acitivities are performed due to the loss of work from the jumped tasktracker (shown in yellow). The intermediate data from the map tasks completed by the jumping TaskTracker is no longer available for the future reduce tasks and thus that work must be repeated. As illustrated in Figures 3.2 (c) and (f), when jumps occur during the reduce phase, map tasks must be re-executed and reduce tasks must begin shuffling intermediate data to complete the MapReduce job.

3.5.2 Performance Measurements

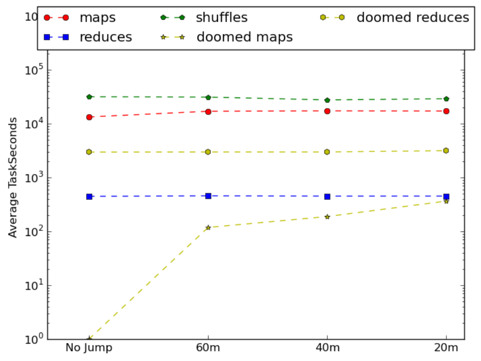
There are two important measures in understanding the performance of a jumping cluster. The first measure, the "effort" overhead, is the overhead of work that must be re-executed because of a jump. The second measure, "performance" overhead, is the additional wall-time needed to complete the job. We record the performance of our experiments and present the results with a focus on these two measures in this section.

Observing the information shown in Figure 3.2, we calculate the effort overhead as the area of the jumped map and reduce tasks. This extra computation is wasted since it must be repeated after the jump. We can quantify taskseconds for all phases of the job by taking the area under the curve of each type of task.

The mean taskseconds for each cluster for both the wordcount and terasort applications are shown in Figure 3.3. Tasks that are executed by a node that is jumping during the job execution are refered to as "doomed" since they are lost and re-executed. The results are interesting in that they show the doomed reduced tasks incurr no additional taskseconds than the Hadoop normal effort overhead of speculative execution of straggler reduce tasks. Furthermore, in both cases of (a) wordcount and (b) terasort, the doomed map taskseconds increase as jump times decrease, but this effort overhead is still two orders of magnitude less than the taskseconds of the successful map tasks.



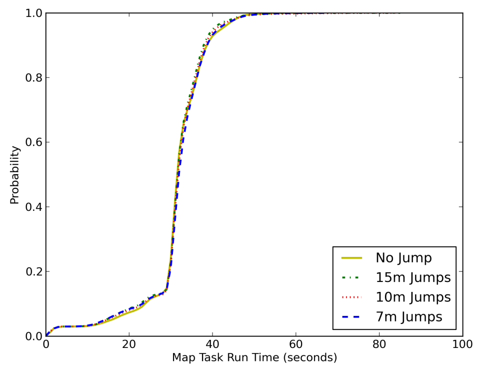
(a) Wordcount TaskSecond Averages



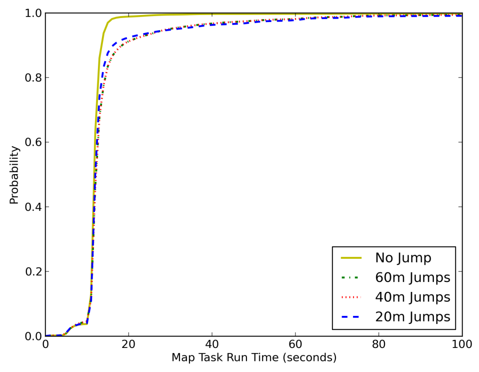
(b) Terasort TaskSecond Averages

Figure 3.3: Map, shuffle and reduce taskseconds and doomed map and reduce tasks for wordcount and terasort jobs under various jump times

The running times of the MapReduce jobs themselves are extracted from the normal logging system of Hadoop. Since our experiments use the same job running on the same static dataset, we have control over many variables that affect the run time of MapReduce jobs. Those variables that are outside of our control are locations on disk where intermediate values and reduce outputs are written, network traffic from adjacent nodes not part of the Hadoop cluster, and variations in task



(a) Wordcount Map Task Time CDF



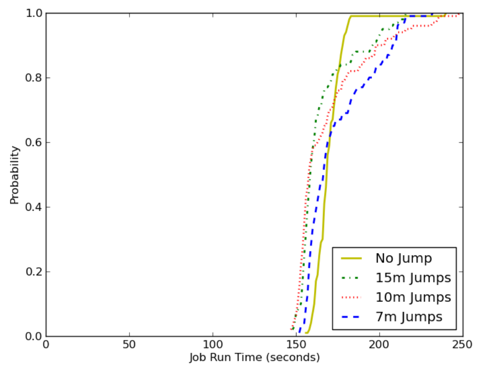
(b) Terasort Map Task Run Time CDF

Figure 3.4: CDF of Map Task times for WordCount and TeraSort

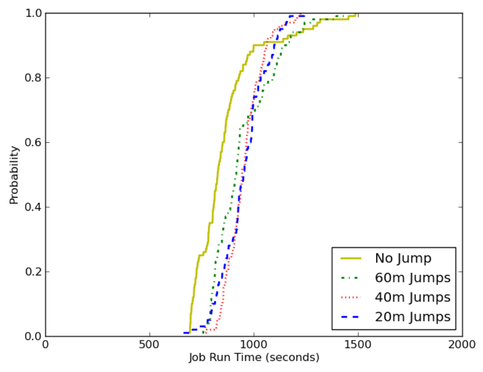
scheduling from the JobTracker.

The map task run time cumulative distribution function (CDF) in Figure 3.4 for both the (a) wordcount and (b) terasort jobs show that jumping has minimal impact on the map task time. This is due to the fact that the system only reports the time for the successful attempt of each map task. For instance, in the case of the wordcount experiment, there are 200 HDFS blocks on which map tasks are placed. If there is a 50% failure rate on map tasks, the time spent during those failed attempts for the 100 map tasks would not be included in the time to complete those tasks. As such, viewing the map task time does not help to show the imposed overhead of a jumping node. That is where information in Figure 3.3 is more relevant.

The job run time CDFs in Figure 3.5 for both (a) wordcount and (b) terasort jobs show the expected behavior of longer run times as the frequency of the jumps increase. Each jump time plot is observed to follow the non-jumping plot up until the point where it reaches the percentage of jobs that are under jump conditions. This is directly related to the ratio of average run time and the jump time for the cluster. For instance, the wordcount job, when not jumping, has an average run time of 3 minutes. When jumping every 7 minutes, 60% of the jobs are unaffected by a jump. Observing the CDF plot shows that the slope of the plot is near vertical up to 60% and the remaining 40% begins to flatten as overhead is observed.



(a) Wordcount Job Run Time CDF



(b) Terasort Job Run Time CDF

Figure 3.5: CDF of Job times for WordCount and TeraSort

3.5.3 Optimizing Jump Time

Due to the observed overhead of JUMMP, a very short jump time is not preferred for a maneuverable Hadoop cluster. As the cluster' size increases and reservation window decreases, it becomes necessary for JUMMP to be configured such that a node jumps before the expiration of its reservation window. An optimal jump time is one that minimizes the overhead of the jumps but also allows the size of the cluster to be maximized for improving parallel processing. Equation 3.1 shows the derivation of the maximum time to jump (t_j) based on reservation window ($RESV$) and size of the cluster (n).

$$\begin{aligned} n \times t_j &\leq RESV \\ t_j &\leq \frac{RESV}{n} \end{aligned} \tag{3.1}$$

Characteristics of the behavior of Hadoop also contribute to the calculation of the minimal jump time. These factors involve data replication speeds and MapReduce job execution timing. We look at each of these individually to discuss how they contribute to minimizing the jump time.

The default replication factor in Hadoop is three. This means that a data block exists within the local storage of three DataNodes. This is configurable by the cluster administrator at start up or on a case-by-case basis when data is written into HDFS. When the NameNode detects a block is under-replicated, it assigns another replication to be stored throughout the cluster. JUMMP initiates this replication pro-actively when a jumping node decommissions itself from the Hadoop cluster, as previously mentioned. If the jump time is so small that nodes are jumping faster than the blocks can be replicated, the JUMMP could become unstable due to missing blocks. The speed at which data is replicated is related to average number of blocks on the node, the speed of the network interconnect, and the workload of the NameNode to reassign block locations. Calculating the minimum jump time as related to data replication is an area of future work.

Finally, optimal jump time is tied to the timing of the currently executing MapReduce job. As shown in Figure 3.2, when a TaskTracker jumps, its currently executing and previously completed tasks in the running MapReduce jobs must be reassigned. The biggest impact is observed when a TaskTracker with an active reduce task jumps. This causes the reduce task to start over and have any unavailable map intermediate data to be recalculated. If nodes are jumping so frequently as to never allow a rescheduled reducer to complete execution, then JUMMP becomes unstable and jobs are interrupted. We thus find that the cluster remains stable in the present of jumps as frequent as the average execution time of the reduce tasks of the currently executing MapReduce job.

3.6 Discussion

In our first research approach to investigating maneuver for resource provisioning, we have presented the Job Uninterrupted Maneuverable MapReduce Platform. JUMMP is an automated scheduling platform that provides a customized Hadoop environment within a batch-scheduled cluster environment. Through its design and redundant nature, JUMMP enables an interactive pseudo-persistent MapReduce platform within the existing administrative structure of an academic high performance computing center. Our experimental evaluation shows that JUMMP can be as efficient as a persistent Hadoop cluster on dedicated computing resources, depending on the jump time. Additionally, we show that the cluster remains stable, with good performance, in the presence of jumps that occur as frequently as the average length of reduce tasks of the currently executing MapReduce job.

Chapter 4

Maneuver for Application Optimization

Our second area of interest for maneuverabilty in cyberspace systems is application optimization. Our research with the Flow Optimized Route Configuration Engine (FORCE) investigates this enhancement. This is an instrumented, representative network testbed where the network topology of a cluster can be maneuvered to develop novel approaches to optimize traffic flow and timing for a datacenter traffic.

The Flow Optimized Route Configuration Engine (FORCE) [63] is an emulated datacenter testbed with a programmable interconnection controlled by an SDN controller. Software-defined networking (SDN) combined with distributed and parallel applications has the potential to deliver optimized application performance at runtime. In order to investigate this enhancement and design future implementation, a datacenter with a programmable topology integrated with application state is needed. The FORCE proceeds us down the path towards this goal. We also utilize Hadoop as a case study of distributed and parallel applications along with a simulated Hadoop shuffle traffic generator. The testbed provides initial experimental evidence of support to our hypothesis for future SDN research. Our experiments on the testbed show a difference in application runtime a factor of over 2.5 times on shuffle traffic for Hadoop MapReduce jobs and the potential for significant speedup in warehouse scale data centers.

4.1 Introduction

Software-defined networking (SDN) combined with distributed and parallel applications has the potential to deliver optimized application performance at runtime. In order to investigate this enhancement and design future implementation, a datacenter with a programmable topology integrated with application state is needed.

Much potential exists in the integration of SDN technologies and performance optimization of distributed and parallel applications within a datacenter. Building entirely new experimental data centers or modifying existing production data centers to study this investigation is costly in time, dollars, and operational output. Historically, researchers have used testbeds or controlled infrastructure to experiment with information technology systems resembling real systems and networks. This approach is advantageous to the SDN and datacenter researcher and can provide significant gains if establish with realistic conditions and instrumented to give constant, measurable results. Our results are the first step to provide a capability that leads to a roadmap for developing methodologies to study this integration.

The contributions of this work include the following:

- The design and prototype development of a datacenter testbed with a reprogrammable network topology. The testbed includes a Virtual Topology Engine that builds virtual network topologies over physical links with SDN flows and a Flow Network Evaluation system to generate a network congestion estimation score.
- The design and development of a Hadoop shuffle traffic simulator designed to place realistic load on a datacenter network.
- Experimental results to show that placement of computation racks within a datacenter topology can have significant impact of the Hadoop shuffle traffic completion time.

4.2 Motivation

In this section, we describe our motivations and goals for the research infrastructure testbed, which includes the enabling capabilities of software-defined networking. Our overall research goal is to investigate technologies and methods for optimizing the performance and energy efficiency of parallel and distributed applications in a cluster or cloud environment. Our goals include the study of how the performance of distributed applications is impacted by the network topology of the datacenter [89]. Analytic and simulation performance models often do not capture all of the characteristics of real applications in a complex distributed environment. At the same time, experimenting with real systems and manipulation of real datacenter topologies can be very expensive in time and dollars. It is not practical to reconfigure an academic production datacenter topology for experimental studies, for example. However, SDN is a technology that can be used to easily and temporarily reconfigure physical and virtual network topologies. Our testbed is a low-cost experimental platform that uses a networked set of single computers and virtualization to emulate the performance of whole racks of machines in a datacenter and their applications. The testbed provides an SDN infrastructure that can be used to study how changes in network topology can impact the performance of the applications.

Several research goals lead us to the integration of SDN and parallel and distributed applications. First, we desire to build a low-cost reconfigurable testbed. We use a modest number of workstations and custom software to emulate the behavior of the entire datacenter. Secondly, we strive to use the testbed to study the effects of network topologies on distributed applications. Our testbed is configured with SDN hardware. Our research goals include using the testbed to gain insight and an early indication of which hypotheses of the use of SDN are worthy of further investigation. Lastly, we would like to understand how execution time modifications of the topology of the datacenter can be used to optimize application performance.

The relationship between network throughput and performance in the Hadoop distributed computing platform has been studied in the literature. Network congestion can delay the movement of data between compute nodes in the shuffle phase of a MapReduce job as reduce tasks fetch data from completed map tasks [89]. These delays can cause overall performance degradation as well as straggler tasks that lengthen the total run time and overuse the available task slots [28]. Network performance also affects operations in the cluster's underlying distributed file system, such as data replication in HDFS [85] and bulk transfers in OrangeFS/PVFS [78].

Congestion in the network can arise for several reasons, such as inadequate provisioning of overall network resources or imbalance in the application workload. Congestion can also occur because of suboptimal decisions or lack of adaptivity in the allocation of the available network resources. In typical cluster configurations, multiple paths exist between any two nodes. Although network switches incorporate forwarding algorithms that attempt to balance traffic across multiple paths, legacy load-balancing algorithms on switching hardware spread traffic based on hashing the source and destination address and protocol [69]. This limits coexistent flows' ability to leverage available capacity in the data center network, causing congestion even when there is available bandwidth in an alternate path. For short flow applications such as MPI the performance can be more sensitive to latency than throughput and the existing mechanisms for network configuration and adaptation may often be adequate. However, in data-intensive applications such as MapReduce and its supporting file systems the workload tends to pass data in very long flows. These types of applications are more tolerant of latency and the predominant limiting factor in performance is throughput. In these cases the performance can be improved by execution time network allocation decisions that are enabled through heightened visibility and maneuverability of the network.

SDN allows a operator-managed controller such as Floodlight [8] with a high level view of the network to remotely program network switches through a protocol such as OpenFlow [60]. The controller is capable of programming its associated switches either reactively or proactively. In the reactive case, a switch receiving a packet for which it has no forwarding rules can query the controller to install rules matching aspects of the packet's header. The controller can also respond to other events, such as a signal of switch failure, to proactively install forwarding rules. Afterward, matching packets are forwarded according to the installed rule.

4.3 Architecture

In this section we describe the architecture design of the FORCE testbed. We discuss the hardware, the off-the-shelf software, and custom software designed by the research team. We introduce our virtual topology engine and our flow network evaluation system. A novel design aspect of the FORCE is the use of single workstations to emulate entire datacenter racks full of computing nodes. This emulation allows us to study the inter-rack networking traffic for distributed applications and to understand how topologies impact performance.

4.3.1 Hardware

The hardware of the FORCE includes one primary server, twelve client workstations, and two SDN-enabled switches. We note that the testbed is extensible and scalable to a very large size. This set of computers used in the testbed is repurposed from upgraded student laboratories and completed research projects and is very low cost. The equipment is installed in a location that allows students to have physical access to the equipment throughout the system building and experimentation.

The primary server is a Dell Precision 330 workstation powered with an 2.40 GHz Intel Core2 Quad processor with 4 GB of RAM. An additional dual-port gigabit Ethernet card was added to the server to provide additional network connections. The twelve client workstations are Sun Ultra20-M2 workstations, each with a dual-core AMD Opteron processor with 2 GB of RAM. Two additional dual-port gigabit Ethernet cards were added to these workstations to provide four additional network connections, bringing the total Ethernet ports to six each.

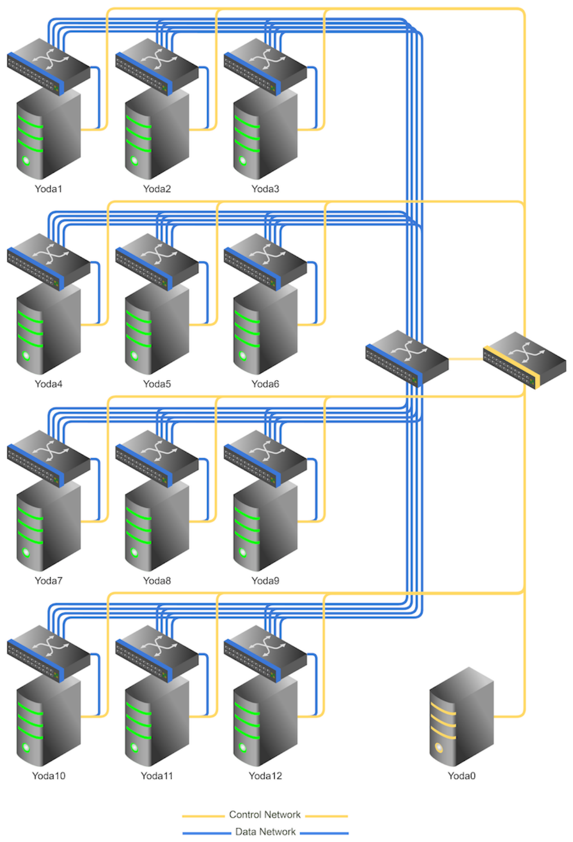


Figure 4.1: The "Yoda" experimental SDN cluster, with separate control and data networks and multiple connections to a OpenFlow-enabled switch.

The two network switches are Pica8 Pronto 3290 48-port gigabit Ethernet switches. Each switch is OpenFlow enabled. Two VLANs are established on one SDN-enabled switch for the `control` and `access` netwoparagraphrks. The server and the workstations each have one connection to both of these VLANs. The remaining 48-port switch is connected to each of the additional four gigabit Ethernet switches of the twelve workstations. This switch allows each workstation to be connected to a maximum of four other workstations. This switch is the primary target of the reprogrammability of the FORCE testbed.

4.3.2 Software

The FORCE package contains a number of important standard and custom software suites and tools for managing the testbed and quantifying experimental results.

Open Source Software The FORCE utilizes several open source software packages without modification, including Ubuntu Linux 12.04.2, Open vSwitch 1.11.0, Floodlight v0.90, and Python 2.7. The server and workstations in the cluster all use Ubuntu Server 12.04.2 as the base operating system. Since the release of Linux kernel version 3.3, Open vSwitch has been the default bridge system. Open vSwitch is used to provide a virtual top-of-rack switch on each of the 12 workstations. This allows each workstation to emulate one datacenter rack with a configurable number of computational nodes. There is a many-to-many communication network between these virtual switches through the 48-port Pica8 physical switch as described above. The server node serves as the central management node of the SDN-enabled testbed, using Floodlight v0.90 as the controller software. The server is configured to control the two physical switches along with the 12 virtual switches. When a new topology is deployed, this node issues the OpenFlow commands to install new flows on the switches. Python 2.7 is core to the operation of the FORCE. Python scripts are used to select topologies, issue OpenFlow commands, start experiments, collect data, and build graphs. The specific software packages are discussed below.

Virtual Topology Engine The core of the virtual topology package is the `force` tool, which implements a virtual network topology by installing forwarding rules on the SDN switches in a cluster. The topology is described in a series of layers using the NetworkX Python package [5]. Each layer maintains a network graph, as well a method for finding a path between any two vertices. The lowest layer describes the physical topology of hosts, switches, interfaces, and links, including characteristics such as hardware addresses and link speeds. The tool then applies subsequent graph layers that abstract each previous layer, building the virtual topology by translating edges into paths on the underlying graph. As a whole, this layered abstraction approach allows us to map desired connectivity among vertices on the highest level graph to the lowest level flow rules to be installed onto the SDN switches.

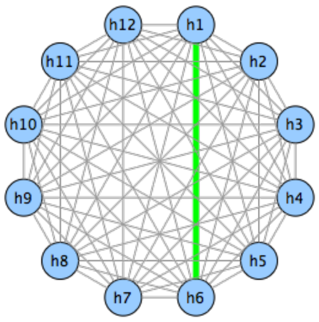
As an example, a virtual 2D-torus topology can be implemented in three layers, as shown in Figure 4.2. In the first layer, the physical network is described. In the case of our experimental cluster, we define a single 48-port switch, 12 virtual switches with 4 links each to the central switch, and 12 hosts with links to the virtual switches. Then, we describe the desired topology, where each virtual switch is connected to its neighbor in a 2D-torus. Each edge in the virtual topology represents a path in the underlying topology, so the edge $vs1, vs2$ in Figure 4.2b translates to the path $(vs1, pronto, vs2)$ in Figure 4.2c. The topmost layer describes the hosts that are logically connected, and each edge describes the full path between those hosts: the edge $h1, h6$ translates to the path $(h1, vs1, pronto, vs2, pronto, vs6, h6)$. Rules are then installed on the SDN switches: for this path, rules are installed to forward traffic on *pronto*, $vs1$, $vs2$, and $vs6$.

Congestion Metric As our software was written to experimentally evaluate network topologies, we wrote a tool to predict the congestion level of the network given a set of source/sink pairs. This tool approximates the mean of all flows' congested flow potentials divided by their flow potentials when free of congestion, if all concurrent flows in the network were in a state of equilibrium. We refer to this metric as the Link Sharing Score (LSS), where $0 < LSS \leq 1$. An LSS of 1 means that there are no overutilized edges in the flow network, and a score of 0.5 would describe a network where congestion cuts flow rates to 50%, on average.

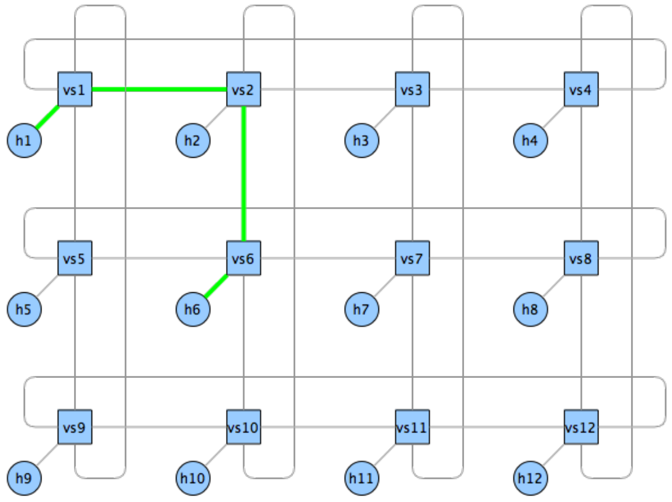
In Algorithm 1, *topo* is an object containing a graph representation of the computer network and a method that computes a static path between a pair of nodes. $pairs$ is a set of source-sink pairs, representing all concurrent flows on the network. In our experiments, this was a list of all mapper-reducer relationships for a set of simulated Hadoop jobs.

The first loop of the algorithm assigns a usage amount to each edge in *topo* by summing the potential rate of all flows through that edge. Line 3 finds the path from *src* to *dst*, which could be the shortest path or any other walk that has been defined. Line 4 establishes the potential uncongested rate of a flow by finding the lowest capacity edge in the path. In lines 5 and 6, this rate is added to the usage of each edge in the path. This may or may not result in an overutilized edge.

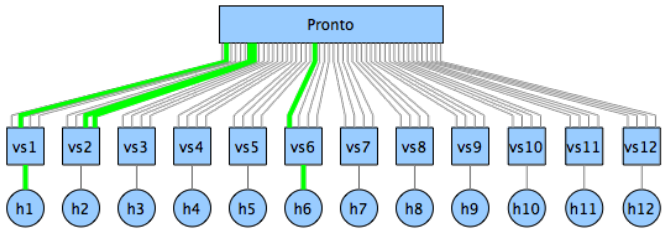
The second loop finds the congested rate of each flow by finding the minimum capacity of



(a) The topmost layer, defining all possible source/destination pairs.

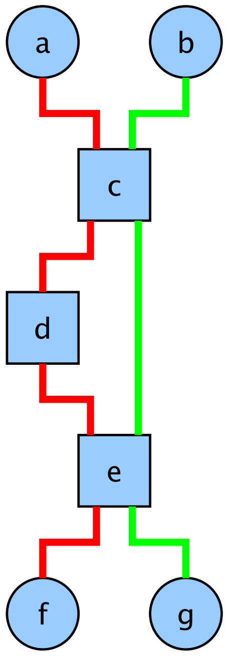


(b) Each abstraction layer defines the paths through the previous layer, as well as a routing algorithm between nodes on this layer.

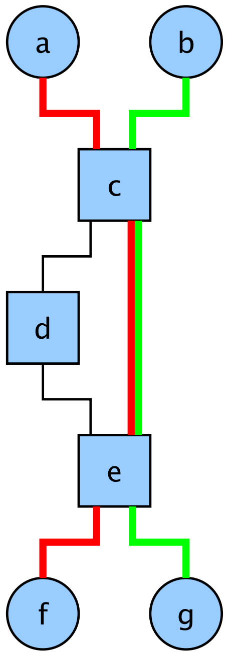


(c) The physical topology layer, consisting of hosts with virtual switches, each connected through 4 interfaces to a central SDN-enabled switch.

Figure 4.2: A virtual topology is described as a sequence of abstraction layers, with each edge representing a path on the layer below it. In this series of diagrams, $h1, h6$ in **(a)** translates to the path $(h1, vs1, vs2, vs6, h6)$ in **(b)**, which then becomes h1, vs1, pronto, vs2, pronto, vs6, h6 in **(c)**. Paths on the lowest level become sets of rules to be installed on switches *pronto*, $vs1$, $vs2$, and $vs6$.



(a) Source/sink pairs (a,f) and (b,g) with paths (ac, cd, de, ef) and (bc, ce, eg).



(b) Source/sink pairs (a,f) and (b,g) with paths (ac, ce, ef) and (bc, ce, eg).

Figure 4.3: In **(a)**, the two paths do not share a link, and the LSS algorithm assigns a score of 1. In **(b)**, edge *ce* is shared, cutting both flows to half capacity and giving a mean congestion score of 0.5.

the congested edges in its path. Lines 10 and 11 again establish the path and uncongested rate along that path. Lines 12-16 find the lowest shared capacity edge in the path, with Line 14 calculating the shared capacity based on the fractional usage of the edge and Line 15 receiving the congested rate. Lines 17 and 19 compute and return the mean quotient of the congested rates to the uncongested rates.

4.4 Experiment

We select Hadoop MapReduce for the representative use case for demonstration of the utility of the FORCE testbed. The literature [89] describes the potential speedup of MapReduce shuffle traffic that is possible by positioning datacenter racks that contain the reduce tasks in close network proximity to datacenter racks that contain the map task from the same MapReduce job.

Algorithm 1 Link Sharing Score

```
1: procedure LSS($pairs, topo$)
2:     for all $(src, dst)$ in $pairs$ do
3:         $path \leftarrow topo.path(src, dst)$
4:         $path\_rate \leftarrow min\_link(path)$
5:         for all $edge$ in $path$ do
6:             $edge.usage \leftarrow edge.usage + path\_rate$
7:         end for
8:     end for
9:     for all $(src, dst)$ in $pairs$ do
10:        $path \leftarrow topo.path(src, dst)$
11:        $path\_rate \leftarrow min\_link(path)$
12:        $rate \leftarrow path\_rate$
13:        for all $edge$ in $path$ do
14:            $scaled\_rate \leftarrow path\_rate \times max(1, \frac{edge.cap}{edge.usage})$
15:            $rate \leftarrow min(rate, scaled\_rate)$
16:        end for
17:        $total \leftarrow total + \frac{rate}{path\_rate}$
18:    end for
19:    return $\frac{total}{len(pairs)}$
20: end procedure
```

This proposed enhancement requirs the dynamic reallocation of point-to-point connections between top of rack switches in a two-dimensional torus topology. Our testbed is ideally suited for testing this optimization. In this section we describe the use of the FORCE to experimentally test how MapReduce performance is impacted by reallocation of point-to-point connections and to determine if further research and investigation of the method is justified.

4.4.1 Hadoop Shuffle Simulator

The nodes comprising the testbed are not robust enough to run a real workload consisting of multiple Hadoop jobs or multiple virtual Hadoop nodes. To solve this problem, and to ensure that the testbed supports a realistic shuffle traffic network load that corresponds to the traffic between datacenter racks, we implemented a Hadoop shuffle traffic emulator within the FORCE. We implemented a centrally controlled software suite that synchronizes bulk network data transfers from map tasks to reduce tasks within the cluster. These data transfers are the same as the movement of map tasks outputs to reducers across the cluster as seen in the shuffle phase of a real MapReduce job.

The Hadoop shuffle emulator reads configurations from a set of files that describe the pseudo Hadoop cluster within the FORCE. The cluster configuration file defines the number of emulated nodes within in each virtual rack, the number of TaskTrackers within each emulated node, the number of map and reduce slots present per TaskTracker, and the default HDFS block size. The workload configuration file defines the number of jobs to be placed on the cluster during an experiment, the input size of the data to be processed by the job, and the type of MapReduce job for each job. With the configuration variables retrieved from these files, various workloads and workload execution scenarios can be emulated within the FORCE without the need for robust servers running multiple Hadoop virtual machines.

Given the set of configurations and experiments, the system is able to deploy the emulated MapReduce jobs across the cluster that perform transfer of shuffle traffic. Based on the size of the data to be processed and the HDFS block size, the system is able to determine the number of maps task required for each job. The number of reduce tasks is determined by a default global parameter or specific argument on a per job basis. The type of job to be run determines the ratio of map task output relative to the block size. After the system is configured with the number of map and reduce tasks per job and the size of the data transfer between all the map tasks and the set of reduce tasks in each job, the transfer of data from map tasks to reduce tasks begins. Since we are only interested in studying the inter-rack traffic, any map and reduce tasks that reside within the same virtual datacenter rack do not transfer data.

In this section we describe the design of the experiments using the FORCE and our Hadoop shuffle simulator that study the impact of placement of reduce and map racks within a datacenter. We describe the cluster configuration, the workload placed on the environment, the data collected, and the method of collection.

We initialized our cluster with the full set of 12 racks and 16 compute nodes per rack, with 8 map slots and 4 reduce slots per node. This configuration emulates a cluster with 1536 map slots and 768 reduce slots. Using the method of bin-packing placement [89], we ran three simulated MapReduce jobs with reducers exclusively on three separate computing racks. Each reducer rack had its own set of three feeding map racks. Thus we had three distinct sets of mapper and reducer racks divided across our datacenter.

In our first experiment, each experimental run consisted of using the FORCE to build and record a random 4x3 two dimensional torus topology, beginning continuous shuffle traffic from each map rack to its destination reduce rack, and recording the total bytes transferred from each virtual top-of-rack switch at each map rack. We ran 1000 experimental runs and recorded the amount of data transferred at each map rack at equally spaced times. These measurements allowed us to compute the average transfer rate at each map.

In our second experiment, each run consisted of again randomly placing the rack in different positions in the 4x3 torus topology. We then began the flow of data between mappers and reducers in deterministic sizes. We recorded the time required to complete each of these data flows. This is different from the first experiment since less congested paths allow the transfer to finish sooner, permitting straggler flows to continue on a network free of congestion.

In both experiments, we established a baseline for comparison with randomized topologies by measuring the results with 500 runs using a fixed network topology with no particular distinguishing characteristics.

4.4.2 Performance Evaluation

In this section we report evaluation results are on the network performance and for the Hadoop MapReduce application.

4.5 Analysis

4.5.1 Network Performance Results

After sampling the mean transfer rate of 1000 random topologies, we found that the mean and variance were significantly different than obtained with 500 runs using the baseline topology, as shown in Table 4.1. We also found that the mean throughput for the baseline topology is 620.7Mb/s while the mean throughput for the randomly set of topologies is 563.6Mb/s. The variance

Table 4.1: Mean Throughputs on Various Node Placements in a 2D-Torus

Node Placement	Trials	$\bar{x}$	s
same	500	620.7 Mb/s	2.27 Mb/s
random	1000	563.6 Mb/s	79.78 Mb/s

of measured throughputs of the baseline is small, only 2.27Mb/s, whereas the variance of measured throughputs of the set of 1000 randomly generated topologies is much larger, at 79.78Mb/s. This indicates that there are topologies with better and/or worse congestion characteristics. We also found that the baseline topology had a mean transfer rate that was higher than that of random topologies. These results provide the motivation for future investigation into optimal topologies for the MapReduce workloads.

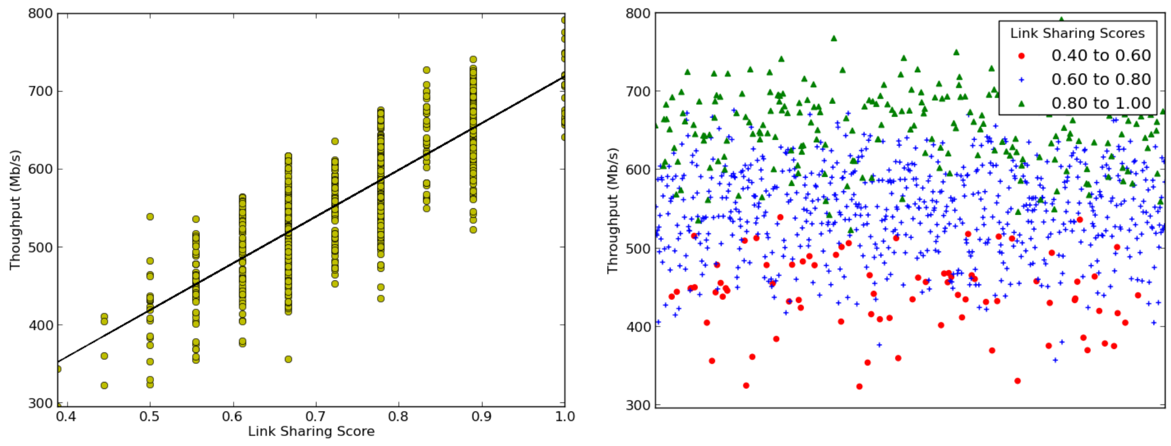


Figure 4.4: The mean throughput of a set of 9 flows is shown, with each point representing a different random placement of nodes within a 2D-torus. The congestion score, with 1.0 representing no shared links, correlates well with the observed throughput.

Next, we used the described LSS congestion scoring algorithm to calculate the theoretical mean congestion of each topology. These scores correlate reasonably well ($r^2 = 0.677$) with the mean throughput of each topology, as shown in Figure 4.4. This tendency can also be seen in Figure 4.4, which supports our conjecture that certain topologies have less congestion than others, given a certain set of flow patterns.

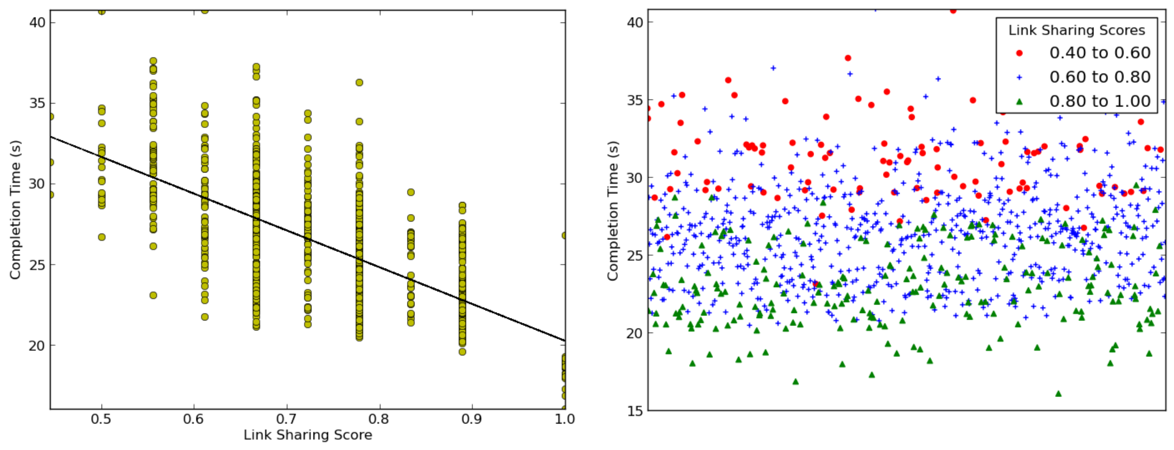


Figure 4.5: Simulated Hadoop job completion time also correlates with network congestion. The difference between optimal and suboptimal configurations can have significant effect on the overall time taken.

4.5.2 Hadoop Performance Results

The results from the second experiment of simulated Hadoop jobs showed similar results. A histogram of simulated job run times is shown in Figure 4.6. As shown in the chart, some run times are over 2.5 times higher than base case times. These worst case configurations ideally would be the topologies an intelligent system would strive to avoid. Additionally, Figure 4.5 shows that the LSS congestion score correlates with the completion time of jobs, which is to be expected as the job completion time affected by the time spent transferring intermediate data. This correlation shows a congestion score is potentially an early indication of shuffle time performance and can be used to intelligently select a topology given a specific network flow pattern.

4.5.3 Implementation Evaluation

The combination of the FORCE testbed along with the Hadoop shuffle traffic simulator has allowed us to obtain encouraging experimental results to support the hypothesis about topological impact on MapReduce shuffle traffic. Without the use of the testbed and traffic simulator, a much more significant investment of research dollars and time would have been needed to obtain experimental results to support our hypothesis.

The experimental results suggest that further investigation is warranted. We are encouraged by the results and can now begin further study of flow optimization under different topologies. Our

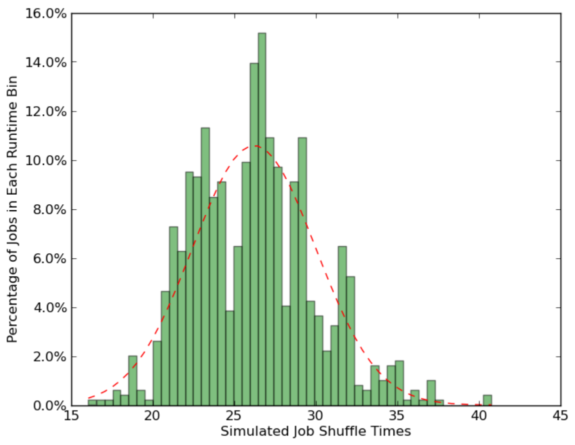


Figure 4.6: Histogram of simulated Hadoop shuffle times with random placement of racks in two-dimensional torus topology.

hypothesis that topology can impact the completion time of MapReduce shuffle traffic is reinforced through experimental results.

The use of the FORCE as an emulated testbed for studying the effect of network topology on application performance in a datacenter has proven beneficial. Continued use of this research methodology could prove financially advantageous to academic and research organizations.

4.5.4 Discussion

In our second research approach of investigating cyber maneuver for application optimization, we have presented the Flow Optimized Route Configuration Engine. The FORCE is a datacenter testbed emulator with a programmable interconnection controlled by an SDN controller. The FORCE allows researchers to get an early indication of the worthiness of data center topology hypothesis. These experimental results come without the cost in time or funding of building production level data centers. Additionally, the system features a Virtual Topology Engine, a Flow Network Evaluation System, and a Hadoop shuffle traffic simulator. We have presented initial experimental results to suggest that datacenter topology, specifically placement within a 4x3 2-D torus network, can impact the time to shuffle intermediate results from a MapReduce job.

Chapter 5

Maneuver for Cybersecurity Improvement

Our third interest area for studying maneuver in cyberspace is improving cybersecurity. Our work with the Defensive Maneuver Cyber Platform [62, 64] introduces a Stochastic Petri Net model of improving the survivability of a distributed and parallel application with the additions of moving target defense and deceptive defense, two of the tactics of defensive cyberspace maneuver [16].

Distributed and parallel systems have become essential operational and logistical tools used in many of today's applications. They are used across academia, military, government, financial, medical, and transportation industries. These systems have garnered the unwanted attention of malicious intruders seeking to disrupt their confidentiality, integrity and availability. Applying the military tactic of cyber maneuver can increase their security and provide defensive mechanisms to extend survivability and improve operational continuity. Understanding the tradeoffs between information systems security and operational performance when applying maneuver principles is of interest to administrators, users, and researchers. To this end, we apply Stochastic Petri Nets to the modeling of security enhancements with the Defensive Maneuver Cyber Platform. This model enables the understanding and evaluation of the costs and benefits of maneuverability in a distributed application environment.

5.1 Introduction

Distributed and parallel systems have become essential tools for the computation of complex problems for multiple sectors of today's society. Government, industry, and academia have made significant financial investments into these infrastructures causing them to become critical to their operational and logistical data driven processes. Protecting these valuable assets while ensure efficient operations is essential to the mission and purpose of the organization.

Malicious actors seeking financial or intelligence gains are targeting supercomputers and distributed computing centers at an increasing rate [2, 26, 70]. Their methods and efforts to disrupt the confidentiality, integrity, and availability of the systems require network security professionals and researchers to invest remarkable amounts of time and money into protecting these assets. We argue that one approach to improving system security is the application of military doctrine and tactics to the defense of distributed and parallel applications. This chapter presents our mode and analysis of of the Defensive Maneuver Cyber Platform focusing on extending the military concept of maneuver to the defense of cybersystems [62, 64].

Security and usability have always been competing interests in information systems. Adding maneuverable elements and features into distributed and parallel applications affects the performance and operational output of the system. Our goal is to understand and characterize how these additional features can improve security while limiting the impact on operations. Our approach is to develop a model of these processes and systems and to use the model for understanding and evaluating these trade-offs.

Our work with the modeling the Defensive Maneuver Cyber Platform (DMCP) provides the following contributions:

- A Stochastic Petri Net (SPN) model of a defensive maneuver cyber platform utilizing moving target defense and deceptive defense tactics.
- An analysis of our SPN model to understand the trade-offs between security and operations in defensive maneuver cyber platform.
- A stochastic model of an attacker for targeting and penetration of systems and how moving target defense and deceptive defense can thwart the attackers success
- Recommendations to augment the SPN model to build prototype systems implementing mov-

ing target and deceptive defense in parallel and distributed applications.

5.2 Architecture

In an effort to provide improved cybersecurity to a distributed and parallel application, we have designed a model of a Defensive Maneuver Cyber Platform using Stochastic Petri Nets. In the model, we represent individual nodes that can operate in specific modes, along with the collection of nodes that constitute the parallel and distributed application. We specifically apply the concepts of moving target defense and deceptive defense to provide increased security. Individual nodes transition between operational, idle, and deceptive modes. Additionally, overall controls of the application seek to ensure that a minimum number of nodes remain operational and deceptive as dictated by the current operational and threat environment. In the event of increased adversarial activity or indications and warnings of threat activity, the system parameters can be adjusted to provide a more secure environment at the cost of operational output. Likewise, in low threat environments, system settings can allow improved productivity at the cost of security. It is understood that these tradeoffs in a system of this type are both necessary and beneficial to system administrators and executive leadership.

In our model, a distributed and parallel system is composed of multiple individual nodes controlled by a separate central management system. We assume that individual nodes are worker nodes of the overall system and can be in one of three states, operational, deceptive, or idle. We further assume that a node does not move directly from an operational state to a deceptive state, but goes to an idle state first, and vice versa. The individual nodes can join or leave the cluster with minimal operational overhead, but otherwise ensure the survivability of the system. One such example of a systems that survives in this manner is JUMMP [65].

Each individual node is comprised of three places (operational, idle, deceptive) and 4 transitions (operational-to-idle, idle-to-operational, deceptive-to-idle, and idle-to-deceptive). There are two universal places (minimum computation, minimum deceptive). There are eight intranode arcs between the places and transitions for each node and an additional six external arc from the control places into the node for management. A single token exists within each node and the place in which it is found determines the state of the node, as described below. The number of nodes is configurable by the system depending on the needs of the application designers.

Formally, a node can be describe as a Stochastic Petri Net as shown in Table 5.1.

5.2.1 Assumptions

We have placed some assumptions on our system to aid in modeling. These assumptions are based on realistic expectations of the system in the real world. We assume that the system has a centralized controller that is not a target of an attacker in this system. The centralized controller is responsible for directing the node transitions between states, monitoring the status of all nodes, and ensuring the rules and parameters of the system are obeyed. We assume the system control is aware of the threat conditions and can make intelligent decisions based on this knowledge. We also assume the system management is willing to lessen the operational throughput when threat conditions exists that increases the probability of adversary action exceeds a predetermined threshold.

Table 5.1: Description of single node SPN

$$P = \{p_o, p_i, p_d\}$$

$$T = \{t_{oi}, t_{io}, t_{di}, t_{id}\}$$

$$\lambda = \{1.0, 1.0, 1.0, 1.0\}$$

$$I^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$I^+ = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$W = \{1, 1, ..., 1, 1\}$$

$$M_o = \{(1,0,0)|(0,1,0)|(0,0,1)\}$$

5.2.2 Individual Node States

When a node is in the operational mode, it is an active member of the parallel and distributed application. In an operational state the node is storing data, processing data, or transferring data to other operational nodes in the system. In the instance of the application Hadoop, these nodes would be engaged datanodes and task trackers performing MapReduce calculations. Operational nodes are listening to known TCP or UDP ports, creating network connections to other nodes listening to these same ports, or transferring data on previously connected flows. Operational nodes are susceptible to attacks from malicious actors, but each is providing an operational contribution as part of the purpose of the parallel and distributed application. Operational nodes, when directed by the central management system, can transition to the idle state. When this transition occurs we assume that the application can continue to operate as a whole working system but with a loss of operational capability.

An idle node is one which is available to be promoted to becoming either an operational or a deceptive mode. In some instances, this node could be part of a larger shared computing resource cluster and available to other users or applications. We assume that idle nodes are non-utilized resources that are available but not susceptible to targeting by an attacker. When directed by the central manager, the node transitions to becoming either an operational node or a deceptive node.

The third mode in which an individual node can exist is the deceptive mode. In the deceptive mode a node is indistinguishable from an operational node to an outside observer. An attacker that scans the network or enumerates the environment will be just as likely to target a deceptive node as it would an operational node. The deceptive nodes are listening to the same TCP / UDP ports, creating connections with other deceptive nodes, and transferring data between each other. Additionally, we assume that decoy nodes have honeypot programs [84] that alert the central management system to the existence of intruders on the network. These triggers can be used to increase the maneuver rate of the nodes or adjust the ratio of computational to deceptive nodes.

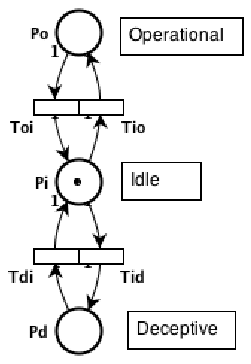


Figure 5.1: Petri Net of individual node which can transition between three modes of operation.

In the Petri Net model, each mode (operational, idle, and deceptive) is represented by a place (P_O, P_I, P_D, respectively). There is a single token which, by its place location, represents the mode of the node. Timed transitions allow the token to move from operational to idle, idle to operational, deceptive to idle, and idle to deceptive. These transitions also are fed by arcs from the centralized control places (P_{No}, P_{Nd}) to ensure the system maintains the minimum number of operational and deceptive nodes. Figure 5.1 shows a single node system with the places labeled with names. The modes for the node are represented on the far right side of the node. The current marking has the token in the idle place, indicating its current state.

5.2.3 System Parameters

The DMCP is composed of one or more individual nodes. The model has a set of configurable parameters that allow exploration of the trade-offs between different system designs. Parameters include the total number of nodes in the system, the number of nodes that are operational, deceptive, and idle in the initial system configuration, and the upper and lower bounds of the number of nodes for each of these states. The parameters can be used to compare tradeoffs of different configurations.

The system has a total number of N nodes. Each node is a single whole node as described above. The system has a minimum number of nodes that must remain in the operational and deceptive states, these are described with the variables O and D. Since Petri Net transitions are atomic and nodes can not transfer directly between operational and deceptive states but must move to the idle state, then $N > O + D$ must hold true for the system in all configurations.

In order to ensure the stability of the maneuver system, the minimum number of operation and deceptive nodes is managed by the system controls with the two places P_{No} and P_{Nd}. These places are initialized with O and D tokens, respectively. An input arc exists between each control place and the transition to the idle state. This arc is weighted to $O + 1$ and $D + 1$ with an output arc back to the control place with weight O and D, respectively. This ensures that a node that transitions into the idle state leaves at least the minimum nodes in its former state, thus not violating the minimal levels set by the system designers. Additional, any transition from idle to operational or deceptive states will deposit a token in the control place. The control place acts as a counter of nodes in its state.

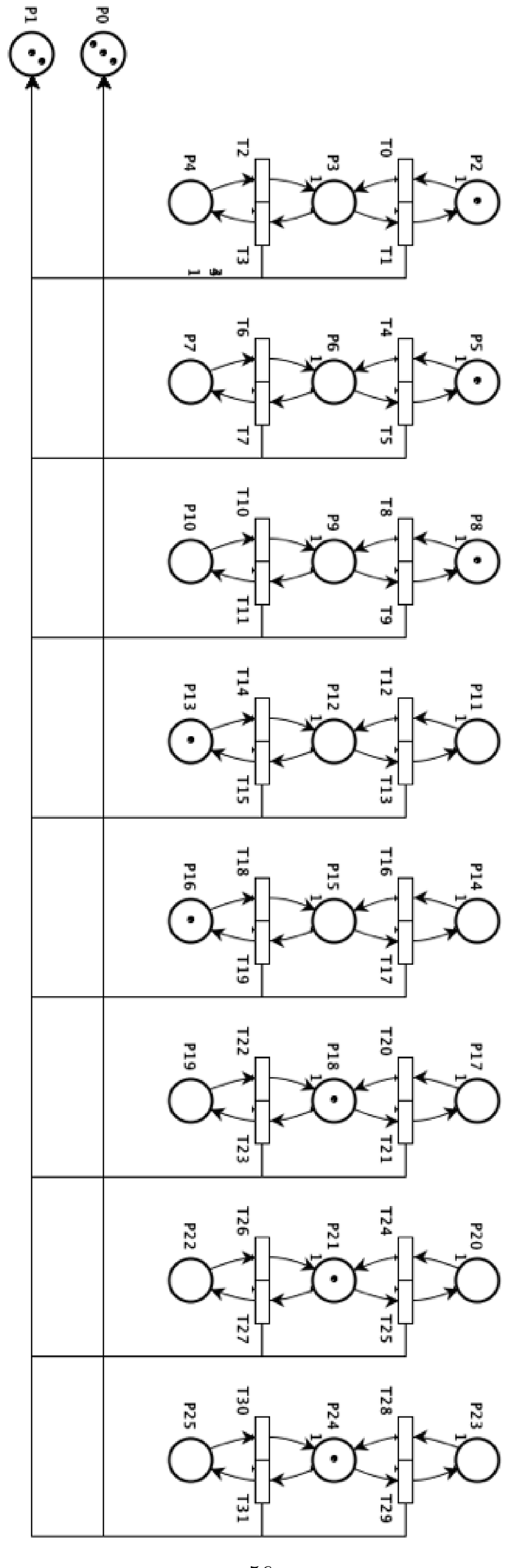


Figure 5.2: An eight node Defensive Maneuver Cyber Platform with three operational nodes, two deceptive nodes, and three idle nodes.

5.2.4 Maneuver Platform Builder

The defensive cyber maneuver system is modeled as a Stochastic Petri Net. We use the Platform Independent Petri Net Editor (PIPE2) [31] to view and evaluate the system. PIPE2 provides a graphical user interface for drawing and building Petri Nets. These nets can then be saved in a Petri Net Markup Language compatible file for use in other PNML tools. PNML is an extension of XML as defined by the standard ISO/IEC 15909 Part 2 [23].

In order to aid in the creation of Petri Nets of our maneuverable system, we designed and developed a maneuver platform builder that automates the creation of the nets. The builder extends the standard XML libraries that currently exist in Python. This custom software tool, called `buildDMCP` takes command line parameters of (N, O, D). The output of the program is a PNML file that can be opened in PIPE2 or any other PNML compatible tool.

The `buildDMCP` tool takes an object oriented approach by defining the building blocks for Petri Nets in general. Functions to build places, transitions and arcs are developed with optional parameters. Using these building blocks, the code builds a single node of our maneuverable system with the respective places, transitions, and arcs. Additionally, the system control are created and connected to the node at the specified locations. Finally, depending on the user provided parameters, the total number of nodes and minimal levels for computational and deceptive nodes are built. The resultant Petri Net has all the places, transitions, and initial markings specified by the user. Additionally, these items are organized and placed on the canvas in a visually appealing manner. Animations and analysis of the nets are done within the PIPE2 application. Figure 5 shows defensive cyber model with $N = 8$, $O = 3$, and $D = 2$ created with `buildDMCP` and visualized in PIPE2. $P0$ and $P1$ in this figure represent the centralized control mechanism of the system (P_{No} and P_{Nd}). These places maintain a count of the current number of operational and deceptive nodes. When a node transitions from idle to either operational or deceptive states, a token is deposited into these places. In order for a node to transition from either operational or deceptive, there must exist $O+1$ or $D+1$ tokens in these places, respectively. In the SPN, this is modeled with output arcs of weight $O+1$ or $D+1$ and input arcs of weight O or D.

5.3 Operational Analysis

To understand the trade-offs between security and operations in our system, we provide some analysis of the DMCP. Understanding the characteristics and behaviors of the system under different configurations will allow system designers and administrators to choose implementations to meet their operational needs. Additionally, this improved understanding of the system will improve our future efforts to build prototype software packages base on this model. We focus our operational analysis on enumerating the potential markings and edges in the reachability graph, studying the different maneuverability states of the system, and analyzing the impact of the transition firing rates on the state distribution probability.

5.3.1 Enumerating and Categorizing State Space

Initially, we want to study how the parameters of the configuration affect the reachability graph of the system. The DMCP is specified with three global parameters, N, O, and D. The current state of the system can be described as a vector C of length N where each element C_j represents the state (S_o, S_d, S_i) of node j^{th} in the system. The total number of nodes in each state can be represented by the variables (o, d, i). For each state or marking with equal counts of nodes in each mode can be in multiple combinations. It can be shown that the total number of states or markings, M, of the Petri Net is defined by Equation 5.1

$$|M| = \sum_{d=D}^{N-O} \sum_{o=O}^{N-d} \frac{N!}{o!\, d!\, i!} \qquad i = N - o - d \tag{5.1}$$

This equation is a summation over the range of two variables that describe the count of computation and deceptive nodes. The equation is similar to the computation of multistate combinations, since with have N nodes that can be either operational, deceptive or idle. The number of operational nodes ranges from the system specified minimum to all remaining nodes after preserving the minimal level of deceptive nodes. Likewise, the deceptive node count ranges from the defined minimal level to all other nodes minus the minimal operational nodes.

The size of the state or marking space of the DMCP is affected by the three variables of the system. It can be shown the maximum state space for a given N is indirectly proportional to the minimum levels of operation and deceptive nodes (O and D). This allows the summation to have a larger range, thus resulting in the maximum state space. The minimal size of the state space is calculated when you maximize O and minimize D or vice versa. This results in only one possible value of o and d in the system.

Figure 5.3 plots the minimum and maximum values of the state space over the range $8 \leq N \leq 128$. As the graph shows, the maximum is on the order of e^n where the minimum is on the order of $\log n$.

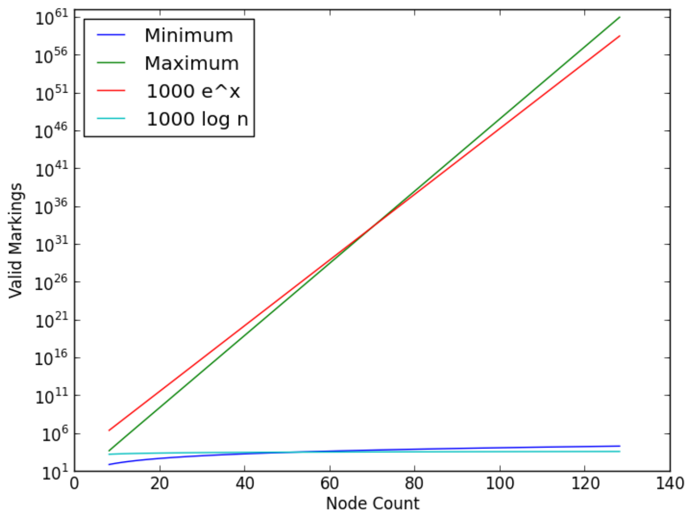


Figure 5.3: Scale of N

The number of edges in the reachability graph can be calculated when consider each marking and how many possible next markings are available. Each idle node can transition to either an operational or deceptive state since all minimums are preserved. If the current number of operational nodes is greater than the minimum number of operational nodes required, each operational node is available to maneuver towards the idle state. Likewise with the deceptive nodes, if the current number exceeds the minimum, those nodes are eligible to maneuver to idle. Using Iverson brackets, where the conditional between the square brackets equals one when the statement is true and zero if false, the number of edges for a particular nodes can be expressed as shown in Equation 5.2. Where o, d, i represent the current number of operational, deceptive, and idle nodes in the system respectively.

$$|E_{node}| = 2i + o[o > O] + d[d > D] \tag{5.2}$$

Calculating the total number of edges in reachability graph can be calculated by multiplying the number of total nodes in the graph (Equation 5.1) by the number of edges per node (Equation 5.2) and dividing by 2. Since reachability graphs are not directed graphs, a pair of markings connected by an edge represents movements in both directions. This product can be shown in Equation 5.3

$$|E_{graph}| = \frac{1}{2} \sum_{d=D}^{N-O} \sum_{o=O}^{N-d} \frac{N!(2i + o[o > O] + d[d > D])}{o!\, d!\, i!} \tag{5.3}$$

5.3.2 Maneuverability States

Each marking of the DMCP can be classified based on the maneuverability of the entire cluster. The maneuverability of the cluster is a measure of the amount that nodes can change from one mode to the next. An idle node is always eligible to change states, where operational and deceptive nodes can only maneuver if the current count of nodes is greater than the minimum levels. When the current state of the system has $o = O$ and $d = D$ the system is considered minimally maneuverable because only the $N - O - D$ idle nodes can maneuver. When the system has more of both operational and deceptive node than the minimum configuration ($(o > O)$ and $(d > D)$) then the system is considered fully maneuverable since all nodes can maneuver at the next moment. Operationally maneuverable states are those where only the operational and idle nodes can be the next to maneuver. Likewise, in deceptively maneuverable states only the deceptive and idle nodes can maneuver.

The number of markings that fall into each maneuver state can also be calculated. Breaking down the number of total markings shown in Equation 5.1, we can build the necessary equations for each set. The minimally maneuverable states are those where the number of operational and deceptive nodes is minimal and are shown in Equation 5.4. The deceptively maneuverable states (Equation 5.5) are those where the number of operational nodes is at the minimum ($o = O$) and the number of deceptive nodes ranges from one more than the minimum ($D+1$) to all but the minimum number of operational nodes ($N - O$). Likewise, the operationally maneuverable states (Equation 5.6) are those where the deceptive nodes are minimized ($d = D$) and the operational nodes range from one more than the minimum $(O + 1)$ to all but minimum deceptive nodes $(N - D)$. Fully maneuverable states (Equation 5.7) are when the operational nodes range from one more than the minimum $(O + 1)$ to all but one more than the minimum number of deceptive nodes $(N - D - 1)$ and all but the current number of operational node $(N - o)$. These equations will be needed when we study the probability of an attacker being able to compromise an operation or deceptive node.

$$Minimally = \frac{N!}{O!D!(N - O - D)!} \tag{5.4}$$

$$Deceptively = \sum_{d=D+1}^{N-O} \frac{N!}{O!d!(N - O - d)!} \tag{5.5}$$

$$Operationally = \sum_{d=O+1}^{N-D} \frac{N!}{o!D!(N - o - D)!} \tag{5.6}$$

$$Fully = \sum_{o=O+1}^{N-D-1} \sum_{d=D+1}^{N-o} \frac{N!}{o!d!(N - o - d)!} \tag{5.7}$$

5.3.3 Transition Firing Rate Impact

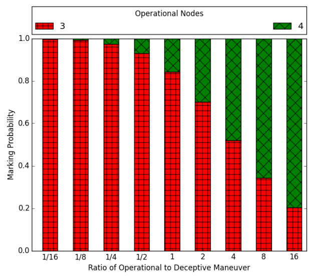
Stochastic Petri Nets have transitions that fire with an exponentially distributed random variable firing delay. Each transition j has a rate λ_j used to calculate the delay. We have studied the effect that the firing rate has on the probability distribution of the steady state of the system.

Since an SPN is isomorphic to a continuous time Markov chain, the steady state distribution π of an SPN can be solved by using the transition rate matrix Q, where Q is the rate between states of the reachability graph and diagonals are the negative sum. Q, also called the infinitesimal generator, has square dimensions equal to the number of markings in the reachability graphs (Equation **??**) and sparsely populated with twice the number of edges in the graph (Equation 5.2). As you can see, as N, O, and D grow the size of Q and complexity to solve for π increases.

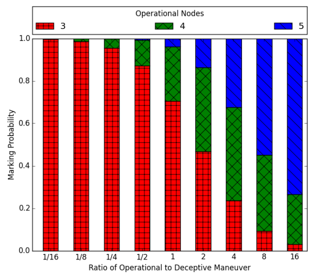
PIPE2 provides a module for determining the reachability graph and conducting the steady state analysis. We use this tool to study the probability distribution for our system and the firing rate impact on the distribution. It can be shown with all transition firing rates equal, the DMCP markings are equally likely.

We began by grouping the transitions in each node into pairs. One group (T_{io}, T_{di}) maneuvers the individual towards operational while the second group (T_{oi}, T_{id}) maneuvers towards deceptive. Varying the rate of the firing rates of the two groups changes the state space probability distribution of the entire cluster. We grouped each possible state by the number of operational nodes (the marking of P_{No}) and calculated their combined probabilities. This way we are able to see how the transition firing rates impact the probability the system has various levels of computational state.

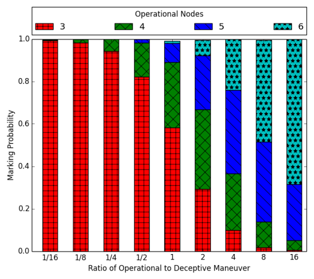
Study the reachability graphs of four separate configurations. In these four configurations, we only varied the total number of nodes (6,7,8,9) while keeping the minimum levels of operational and deceptive nodes consistent $(O = 3, D = 2)$. As shown in Figure 5.4 when decreases the operational maneuver rate with respect to the deceptive maneuver rate increase the likelihood our system will hover around the minimal level of operational nodes. While increasing the operational rate will cause our system to trend towards overly operational. Though as we increase the size of N with each graph we change the highest level of operational nodes, our system still trends towards this highest level.



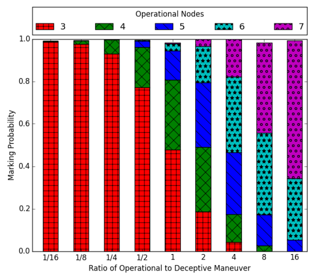
(a) 6-3-2



(b) 7-3-2



(c) 8-3-2



(d) 9-3-2

Figure 5.4: Maneuver rate effect on probability of computational composition

5.4 Security Analysis

In this section, we study the security improvements gained from the addition of moving target and deceptive defense mechanisms of the DMCP. The overall security posture of a distributed and parallel system along with operational considerations allows system designers to make decisions on the number of nodes necessary for deploying the system to meet the operational and security goals of the organization. We will discuss the model of an adversary that we will consider for this analysis along with how each parameter affects the probability of withstanding the attempts of the attacker.

5.4.1 Attacker Model

The model of an attacker that we use to evaluate the DMCP is a single adversary who will target only operational or deceptive nodes. The attacker is unable to distinguish between operational and deceptive nodes and selects with equal probability a random active node to target. Once selected, the attacker targets and attempts to gain user or elevated access to the node using its various on-hand exploitation tools. After a random amount of time being targeted, the node is compromised. We refer to this time as compromise time (t_c) and model it as an exponential random variable with a rate referred to as the attacker compromise rate (λ_{ac}) and mean of $\frac{1}{\lambda_{ac}}$.

The deceptive defense mechanisms of the system protect against this attacker with the introduction of deceptive nodes in two manners. First, the current composition of the cluster with a number of operational and deceptive nodes impacts the probability that a randomly selected active node is an operational node. Secondly, once a targeted node is penetrated, if the node was an operational node, the node is compromised and provides no operational use to the computational purpose of the distributed system. When a deceptive node is penetrated, the embedded honeypot software alerts the system administrators of the presence of an intruder. This notification allows the system administrators and network security personnel to continue to observe the attacker's behaviors or adjust the defensive posture of the system as a whole. The compromise of a deceptive node does not impact the operational state of the distributed system since the node was already in deceptive mode, providing only security enhancement.

Additional protection is provided by the moving-target defense mechanism of the system. As determined by the firing of the transitions to maneuver individual nodes between modes, targeted nodes can maneuver to an idle state before the attacker is able to gain access to the system. An operational node targeted by an attacker can maneuver to idle and evict the adversary from that node, thus interrupting its opportunity to compromise the node. This moving-target defense can also weaken the strengths of the deceptive defense when a targeted deceptive node maneuvers towards idle because it prevents the system administrators from discovering the presence of the intruder. Once ejected from a targeted node, the attacker begins to conduct information gathering and enumeration of the system to select the next node to target.

A summary of the system and attacker parameters is provided in Table 5.2

Table 5.2: System and attacker parameters

System Parameter	**Description**
N	Number of nodes
O, D	Minimal level of operational and deceptive Nodes
o, d	Current number of operational and deceptive nodes
λ_{om}, λ_{dm}	Operational and Deceptive maneuver rates
Attacker Paramter	**Description**
λ_{ac}	Attacker compromise rate
t_c	time to compromise currently targeted node

5.4.2 Attack Survival

We evaluate how the DMCP improves the security of a distributed and parallel system. We will study an overall probability of survival ($P_{survival}$) which we define as the probability that the actions of an attacker targeting a randomly selected active node will not result in the compromise of an operational node. For a baseline comparison, we will assume that a standard distributed and parallel system will have a probability of survival equal to zero. Though this is not quite accurate, any external defenses and security measures employed in support of a standard system could be deployed to our system, so we evaluate how our system increases this probability. Our goal is to attempt to maximize the probability of survival.

Equation 5.8 breaks down the probability of survival into three other probabilities. We discuss the overall equation and then study each individual probability with the goal of reaching a maximum overall probability. The first probability to consider is the probability that a deceptive node is targeted. If the deceptive node is targeted, it cannot result in the compromise of an operational node. This is reflected in (5.8) as P_{dt}. An operational node is targeted with probability $(1 - P_{dt})$. This could result in compromise if the node does not maneuver into the idle state before the node is defeated. The probability of the node maneuvering is a product of the probability that it can maneuver (P_m) and the probability that the maneuver time is less than the defeat time ($P(t_m < t_c)$).

To maximize the overall probability of survival, we want to maximize each individual probability. In the following sections, we discuss the control and configuration parameters of the system and their impact to each of these individual properties.

$$P_{survive} = P_{dt} + (1 - P_{dt}) * P_m * P(t_m < t_c) \tag{5.8}$$

5.4.3 Targeting Deceptive Nodes

The deceptive defense mechanism of the DMCP creates false targets on which we want the adversary to waste valuable time and effort. We desire to maximize the probability that a randomly selected active node will be deceptive instead of operational, while maintaining the required level of operational capability to meet overall system productivity goals. At any given time, this probability, assuming uniform probability of node selection, is number of deceptive nodes divided by the number of active nodes (operational and deceptive nodes). When the number of nodes are equal, this probability is 0.50. We have shown in Section 5.3.3 how adjusting the operational and deceptive maneuver rates relative to each other can impact the operational composition of the DMCP. Using this same analytic approach, we can show how varying the maneuver rate ratios can impact the probability that a randomly selected node is deceptive. Shown in Figure 5.5 is plot from the same steady state analysis of four different DMCP configurations with varying numbers of total nodes (6,7,8,9) but same level for the minimum operational and deceptive nodes ($O = 3, D = 2$). As the Figure 5.5 shows, when the operational maneuver rate is smaller than the deceptive maneuver rate, the size of the cluster and the deceptive node selection probability are positively correlated. When the operational rate is larger than the deceptive rate, the cluster size and probability are negatively correlated. Generally speaking, decreasing the ratio of the operational maneuver rate and the deceptive maneuver rate increases to probability of an advisory targeting a deceptive node rather than an operational node. As discussed earlier improves the probability of survival.

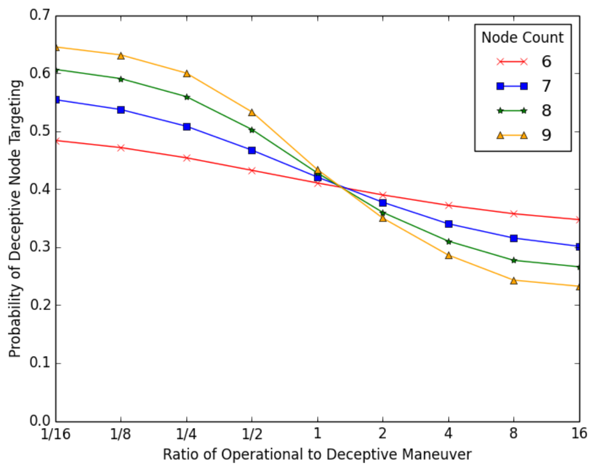


Figure 5.5: Probability of randomly targeting deceptive node as function of maneuver rate ratios

5.4.4 Probability of Maneuverability

Understanding the probability that an operational node can maneuver is related to the maneuverability state. For an operational node to be enabled for maneuver to the idle state, the system must be currently in either the operationally maneuverable or the fully maneuverable state. It can be shown using the steady state analysis, that when the operational and deceptive maneuver rates are equal that the probability of being in any of the possible markings is equal. Equation 5.9 shows the calculation for the probability that operational nodes are enabled to maneuver to the idle state when maneuver rates are equal. This is solved by dividing the sum of the number of fully operational markings (5.7) and the number of operationally maneuverable markings (5.5) by the total number of markings (5.1).

$$P_m = \frac{\sum_{d=O+1}^{N-D} \frac{N!}{o!D!(N-o-D)!} + \sum_{o=O+1}^{N-D-1} \sum_{d=D+1}^{N-o} \frac{N!}{o!d!(N-o-d)!}}{\sum_{d=D}^{N-O} \sum_{o=O}^{N-d} \frac{N!}{o!\,d!\,(N-o-d)!}} \tag{5.9}$$

To determine how N, O, and D affect this probability, we calculate the impact on the probability as we increase the size of the cluster and the scale of the minimum numbers for operational and deceptive nodes as percentages of the overall size of the cluster. Figure 5.6 shows this analysis. In Figure 5.6 we vary the size of the cluster from 16 to 256 nodes while testing different percentages for the minimums. In all calculations the minimums for the operational and deceptive nodes are equal. In the Figure 5.6, the orange line marked with dots shows the result when O and D are equal to 25% of the total number of nodes.

As shown in Figure 5.6, a network where O and D are equal to 37% of N, the probability of an operational node being able to maneuver approaches approximately 60% as N grow very large. A network with minimum levels equal to 43% of N, we will not ever exceed approximately 25% operational node maneuverability. These configurations for DMCP systems would not be optimal when a high level of maneuverability is desired.

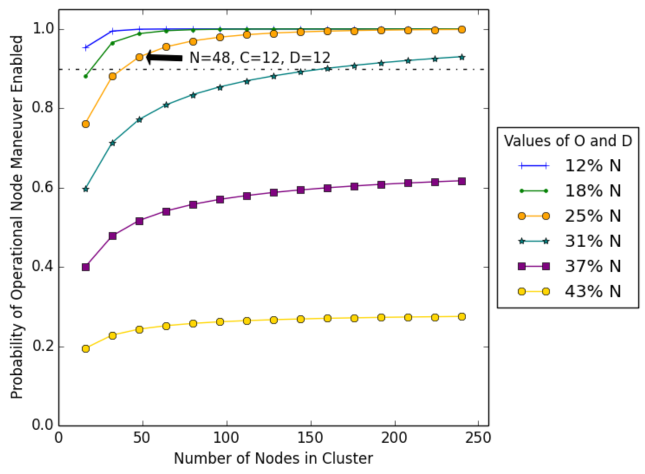


Figure 5.6: Probability of operational node maneuver as function of cluster size and minimum levels

This chart provides rules of thumb that can be used by a system administrator for designing a DMCP given the operational and security goals of the system . For example, the arrow in Figure 5.6 points to the datapoint where $N = 48$ and $O = 12$ and $D = 12$. At this configuration, the probability is greater than 90% that a targeted operational node will be enabled to maneuver. For a goal of 90% maneuverability and 48 nodes are available, this is one possible configuration. Increasing the number of minimal nodes required decreases the probability of an operational node being able to maneuver.

Further analysis of the configuration where $N = 48$ is presented in Figure 5.7. In this chart, we plot the operational node maneuver probability as a function of the number of deceptive nodes with the total number of nodes fixed at 48. The labels on the curve mark are referred to as the knee in the curve. This is the point where the probability of the an operational node being able to maneuver begins rapidly decreasing as you increase the minimal number of deceptive nodes. This is calculated by find the point in which the slope is smaller than -1. We annotate this point as the highest value for the deceptive nodes minimum where the probability remain approximately at its max value. For example, for the plot where the minimum operational nodes is 16, the knee is the curve is where the minimum number of deceptive nodes is equal to 12. A system designer can choose any value for D less than this point and maintain essentially the same probability that operational nodes are able to maneuver when targeted by the attacker.

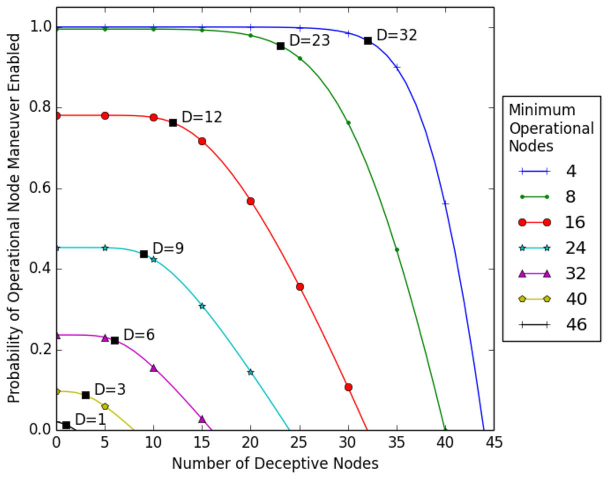


Figure 5.7: Probability of operational node maneuver enablement as function of number of deceptive nodes minimums (48 total nodes)

5.4.5 Maneuvering before Node Compromise

Finally, we strive to maximize the probability that our targeted operational node maneuver before the attacker can successfully gain access to the node, while maintaining the required level of operational capability to meet overall system productivity goals. This is calculated as a probability of the next maneuver time is smaller than the time to defeat the node. Both of these times are modeled as exponential random variables. The probability that of one exponential random variable is less than another random variable is well studied [79] and is equal to the ratio of the first rate to the sum of two rates. For our system, the two rates are the deceptive maneuver rate (λ_{dm}) and the attacker compromise rate (λ_{ac}). Equation 5.10 shows this calculation.

$$P(t_m < t_c) = \frac{\lambda_{dm}}{\lambda_{dm} + \lambda_{ac}} \tag{5.10}$$

When the rate of deceptive maneuver and the rate of attacker compromise are equal, the probability of the maneuver occurring before the compromise is 50%. Figure 5.8 shows how varying the two rates over the range of (0,16) impacts the probability of maneuver before compromise. As the ratio of deceptive maneuver to the attacker comprise rate increases, the likelihood of survivability increases as shown in Figure 5.10. Maximizing this probability increases the overall probability of survival of the system.

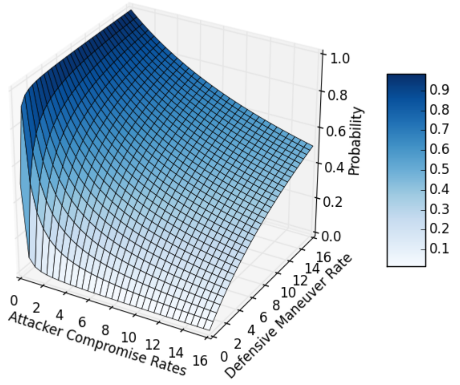


Figure 5.8: Probability of operational node maneuvering before attacker can penetrate

5.5 Discussion

In this section we provide a summary of the impact of the operational and security parameters on the system. By providing a list of rules of thumb for potential implementation of the

DMCP, we give system administrators an initial starting point for designing implementations of our model in prototype deployments. We also address other potential security benefits that have not been addressed in this work.

Maximizing operational capability is essential to an organization that utilizes distributed and parallel computing for computational analysis. System designers and engineers want to ensure they set their minimum operational node count (O) to meet the lowest acceptable output from the system. This value also needs to ensure the cluster remains stable and uncorrupted at this minimal level. As shown in Figure 5.4, it is very possible for your system to spend a significant amount of time at the minimum operational level depending on the values of your maneuver rates. As shown in Figures 5.6 and 5.7, the lower the minimum value for the operational nodes, the higher your probability of your operational nodes being able to maneuver. This brings us to our first rule of thumb:

Rule of Thumb #1: Set the minimal operational node parameter just above the operational breaking point of your system

Deceptive nodes allow attackers to waste time and effort on penetrating decoy nodes allowing operational nodes to be left uncontested and increasing the time to compromise of operational nodes. Additionally deceptive nodes when running honeypot software exposes the presence, interest, and tools of an attacker. We also have shown that the higher the minimum level of deceptive nodes in a system decreases the available nodes that can maneuver between states creating static environments. Static environments defeat our moving target defense advantages and lowers the probability that operational nodes are enabled to maneuver when targeted by an adversary. Figure 5.7 highlights that at some point increasing the minimum level of deceptive nodes causes a significant drop in the probability of maneuverability enablement of operational nodes. In light of these observations, we present our second rule of thumb:

Rule of Thumb #2: Set the minimal deceptive node parameter as equal to or just less than the minimum number of operational nodes

The ratio of the operational and deceptive maneuver rates greatly impacts the composition of your cluster. As shown in Figure 5.6, increasing the operational maneuver rate as compared to the maneuver rate makes the system more likely to maintain a high count of operational nodes.

Conversely, by lowering the operational maneuver rate with regards to the deceptive maneuver rate, the system takes on a higher level of deceptive nodes. Adjusting the ratio of the operational to deceptive maneuver rate is much more preferred method of controlling the composition of the DMCP than adjusting the minimal levels of operational and deceptive nodes. This ratio can be visualized as control knob to be adjusted based on the current threat condition of the system. Under low threat, increase the operational maneuver rate and under high threat conditions, decrease the rate. This is summarized in our next rule:

Rule of Thumb #3: Increase the availability of operation nodes in the DMCP by increasing the operational maneuver rate. Maneuver the DMCP to a more deceptive and defendable state by decreasing the operational maneuver rate.

Maneuvering nodes between the operational and idle modes imposes overhead associated with replicating data blocks, starting and stopping any worker threads, and joining and exiting the cluster membership. The time to complete the overhead tasks delays the beginning of the operational use of the node. Frequent maneuvers into and out of operational mode could cause the system to become unstable or ultimately unavailable. System administrators must understand the threshold frequency where the system becomes invalid and attempt to maintain operational maneuver rates lower than this value. This is captured below:

Rule of Thumb #4: Ensure the operational and deceptive maneuver rates are limited to ensure the underlying distributed and parallel system does not break from having nodes maneuver too frequently

In the event that an attacker targets an operational node and attempts to gain access, the moving target defense mechanisms protect the system by maneuvering to the idle state before the compromise is complete. The probability of the maneuver is a function of the deceptive maneuver rate and the attackers compromise rate. Security professionals should seek to gain intelligence about potential adversaries and understand their capabilities to estimate their rate of compromise. Maintaining a deceptive maneuver rate above the estimated attackers rate will create a better than 50% probability that the node would maneuver before the attack is successful. The last rule of thumb presents this observation:

Rule of Thumb #5: Set the deceptive maneuver rate higher than the estimated attacker compromise rate

There are additional security facets that can be considered for an even further analysis of the DMCP. Currently our analysis is limited to a single attacker. Multiple attackers or multiple simultaneous attacks could be modeled in a similar manner as we have shown here. There would need to be an additional statistical distribution for the number of ongoing attacks factoring into the calculation of the probability of an operational node being compromised. Additionally, a different or more complex attacker model could be used to study how these security enhancements would improve the defensible of a distributed and parallel system. Lastly, we don't consider the residual effects of the presence of an attacker after a node maneuvers to another state. These effects will increase the success rate of future attacks by leaving backdoors or known exploits. Our model also has no mechanism to deal with the increasing dangers of insider threats.

When we consider the targeting of a node running in the deceptive mode, we can use the same analysis from studying the probability of an operation node maneuvering to the idle state before an attacker can successfully penetrate the node. It is advantageous to the system administrator and network security professional to have an intruder successfully gain access to a deceptive node. This provides an indication of an adversary interest in your network and systems, allowing proactive defensive measures to be taken. The probability of success against a targeted deceptive node is similar to (5.10) but replacing the deceptive maneuver rate with the operational maneuver rate (λ_{om}). As such, having an operational maneuver rate lower than the compromise rate would increase the likelihood of the honeypot software detecting the presence of the attacker.

Lastly, the security enhancements from moving target defense in the DMCP only consider when nodes are being targeted. Our model of an attacker also must implement a reconnaissance or enumeration stage to collect data on our system before targeting. Additional security enhancements are provided by the moving target defense mechanism during this enumeration stage. The information gathered is only valuable until the next node maneuvers. Evaluating the enhancement of the MTD mechanisms during this stage is another area that can be expanded upon to have a better understanding of the value of our model on distributed and parallel system security.

In this chapter, we present research into using cyber maneuver for security improvement with the introduction of our model of the Defensive Maneuver Cyber Platform utilizing Stochastic Petri Nets. By utilizing moving target defense and deceptive defense tactics, we have shown that we can increase the complexity and composition of a parallel and distributed application by increasing the ratio of deceptive to operational nodes and increasing the rate in which nodes transition between states. We have theorized these characteristics can present a more defendable platform due to the changing attack surface. We have introduced a stochastic process for an attacker attempting to target and penetrate our system and how the additional security concepts improve survivability and impede the attacker's success. We have presented an analysis of the probability of surviving the actions of the attacker based on the ratio of the maneuver rates and the attackers compromise rate. Lastly, we present rules of thumb to be used by system designers whom wish to implement this model in prototype systems for further evaluation.

Chapter 6

Prototype Resource Maneuver Manager

The final area of our research focuses on combining many pieces from our preceding chapters to build a Prototype Resource Maneuver Manager. This software implementation combines the resource provisioning aspects discussed in Chapter 3 along the security enhancements of Chapter 5. Fortified, the Prototype Resource Maneuver Manager, shows how various aspect of military maneuver combined together can be used to build real maneuverable applications to provide enhancements to distributed computing. Additionally, Fortified verifies the Petri-Net model for a moving-target and deceptive defense system as discussed in Chapter 5.

6.1 Introduction

Modeling of systems provides a mathematical understanding of the behaviors and characteristics expected of implementations. The model allows a designer to verify that the behavior is consistent with the goals and objectives of the system. Modeling is useful when the complexity of the system is such that full mathematical evaluation is unfeasible. Distributed and parallel systems with large amounts of asynchronicities and concurrency are examples of complex system. When building a system based on a model, studying the implementation of the system can be compared to anticipated results based on the model for further understanding and validation.

Our works in designing, building, and modeling maneuverable applications are combined in this chapter to build the Prototype Resource Maneuver Manager. The system, Fortified, is a prototype implementation of the system described with the Petri Net model introduced in Chapter 5. Additionally, we repurpose parts of the underlining code and functionality from JUMMP as discussed in Chapter 3 to deploy this system into the Palmetto Cluster at Clemson University. The prototype system is studied and used to validate the modeling results of moving-target and deceptive defenses as introduced in the Defensive Cyber Maneuver Platform.

This work with the Prototype Resource Maneuver Manager, Fortified, provides the following contributions:

- A fully functional prototype software suite to deploy maneuverable Hadoop clusters into distributed environments with individual nodes that maneuver between operational, idle, and deceptive modes of operation,
- A discrete event simulator version of the Defensive Maneuver Cyber Platform that allows long running simulations to provide statistical analysis of the behaviors of the modeled system, and
- A validation of the mathematical probabilities of survival when targeted by a modeled attacker and a verification of the security enhancements provided by moving-target and deceptive defense mechanisms.

6.2 Architecture

The Prototype Resource Maneuver Manager, Fortified, is a software package built in Python that deploys a working system implementing the resource provisioning aspects of JUMMP along with the moving-target and deceptive defense security enhancements of DMCP. We use Hadoop as the underlying distributed system for both the operational and deceptive nodes. Fortified includes a robust controller that deploys a maneuverable application where client nodes maneuver between the states of operational and deceptive Hadoop client nodes and an idle available state.

6.2.1 Fortified Controller

The main component of Fortified is a multithreaded resource scheduler which establishes the distributed application and maneuvers nodes between states based on the Petri-Net model of the DMCP. Fortified maintains complete knowledge of the state of the each individual nodes and triggers the signals for each node to perform the next maneuver.

Fortified runs in two overall modes of execution: prototype and simulation. In prototype mode, Fortified connects to real distributed nodes in a cluster and starts and stops actual distributed computing system processes and honeypot software suites. Maneuvers occur in real time and allow systems to run for days or weeks while performing real operational jobs. In simulation mode, Fortified acts as a discrete event simulator useful for studying the behaviors of the system to ensure correctness of configurations. Simulation mode also provides verification of the Petri Net model of the Defensive Maneuver Cyber Platform.

Using an object-oriented programming approach with a Model-View-Controller design, Fortified has a list of objects representing each individual node that can maneuver between operational, idle and deceptive states. Each node object has a field for the current mode, the operational maneuver timer, and the deceptive maneuver timer. The operational maneuver timer expires when a node maneuvers towards the operational mode (deceptive to idle or idle to operational). Likewise, the deceptive maneuver timer is for maneuvers towards the deceptive mode (operational to idle or idle to deceptive). Operational and deceptive maneuver timers are set to infinity when the node is already in the operational or deceptive states, respectively. Each maneuver timer is based on the system-wide operational and deceptive maneuver rates as described by the DMCP system. Additionally, the node object possesses the needed functionality to conduct all maneuvers and interact with the controller object. This includes responding to queries from the manager object on current status and timers, updating the current mode, and resetting maneuver timers. The node object also maintains a path to current programs to execute by the actual node to conduct each type of maneuver, such as starting and stopping disturbed system daemons or honeypot software.

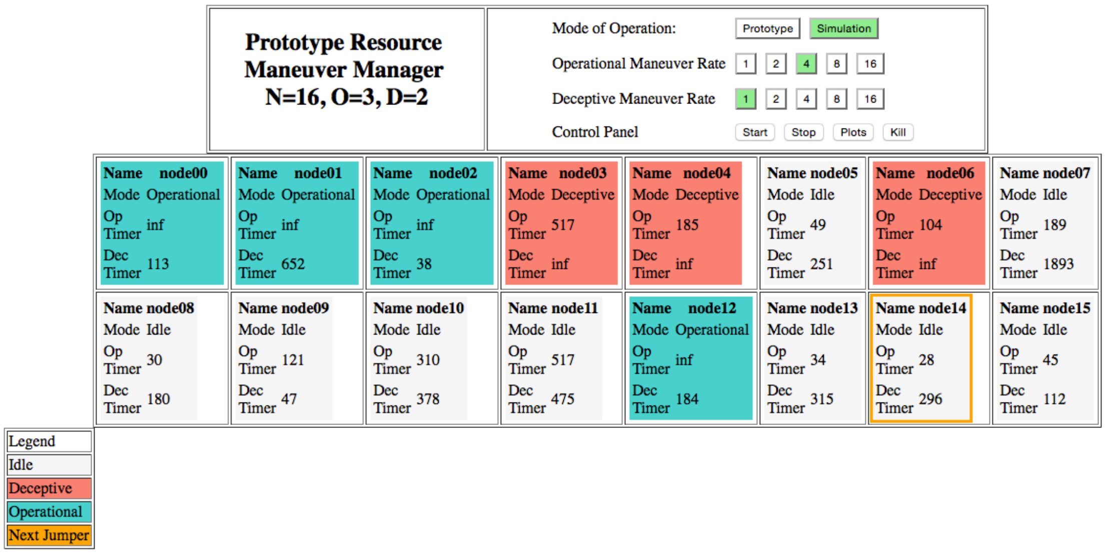
Fortified maintains current global settings for the system. These settings include the operational and deceptive maneuver rates, the minimum number required of operational and deceptive nodes, the overall mode of execution (prototype or simulation) in which Fortified operates, and a detailed event history of all maneuvers. Fortified contains internal functionality to provide an up-to-date view of all nodes and their current mode of operation, either operational, deceptive, or idle. The system is also able to signal individual nodes to maneuver when the timers expire or to reset their timers in the event of change of overall maneuverability status. The system is able to transition between modes of execution or to change the maneuver rates during run time.

The system queries each node to discovery its next maneuver. Based on the earliest maneuver possible between all nodes and the current count of nodes in each state, the system is able to decide if any other timers must be maximized to preserved the system required minimum levels. When the system transitions from fully maneuverable to operationally maneuverable, the deceptive timer for all operational nodes is maximized. This is to ensure no operational node becomes idle which would result in having less than the minimum amount of operational nodes. Likewise, for transitions between fully maneuverable and deceptively maneuverable states, the operational timer for the deceptive nodes is maximized. This is equivalent to the T_{oi} and T_{di} transitions being disabled when P_0 and P_1 reach the minimum level of tokens as shown in Figure 5.1. The reverse transitions (operationally to fully and deceptively to fully) will cause these previously maximized timers to be reset in accordance with the current maneuver rates.

6.2.2 Web-based Interface

A web-based system interface is provided in the Fortified system in addition to the command-line interface. The system interface allows an administrator to see a snapshot of the system, including each node and its mode of operation, current timers, and the next node to maneuver. The interface also provides the ability to start or stop the maneuvering, adjust the operational and deceptive maneuver rates, and change the mode of execution between prototype and simulation. Additionally, the interface provides historical plots of the cluster that can be dynamically built which display maneuvers of each node and the overall trends of the system. A sample snapshot of a 16-3-2 system is shown in Figure 6.1

The plots provided by the controller can also be built with a command line interaction with the system. The first plot is an individual Gantt chart for each node indicating when it was in each mode of operation during the lifetime of the system. The second chart is a stacked area chart plotting the same information but the Y-axis is sorted by node types so total quantity of nodes in each mode can be easily observed. A sample set of charts for the same run of a 16 node system is shown in Figure 6.2 with dashed vertical lines indicating when the operational maneuver rate was changed, which shows the impact on the composition of the cluster. In Figure 6.2, a total of 100 maneuvers are made between each rate change to allow the system to stabilize at the current rates. The system starts out with equal operational and deceptive maneuver rates (1.0). The first dashed vertical line indicates a transition of the deceptive maneuver rate up to 4.0 with the operational maneuver rate



Prototype Resource Maneuver Manager N=16, O=3, D=2

Mode of Operation: Prototype [✓ Simulation]

Operational Maneuver Rate 1 2 [✓ 4] 8 16

Deceptive Maneuver Rate [✓ 1] 2 4 8 16

Control Panel Start Stop Plots Kill

Name	Mode	Op Timer	Dec Timer
node00	Operational	inf	113
node01	Operational	inf	652
node02	Operational	inf	38
node03	Deceptive	517	inf
node04	Deceptive	185	inf
node05	Idle	49	251
node06	Deceptive	104	inf
node07	Idle	189	1893
node08	Idle	30	180
node09	Idle	121	47
node10	Idle	310	378
node11	Idle	517	475
node12	Operational	inf	184
node13	Idle	34	315
node14	Idle	28	296
node15	Idle	45	112

Legend

Idle

Deceptive

Operational

Next Jumper

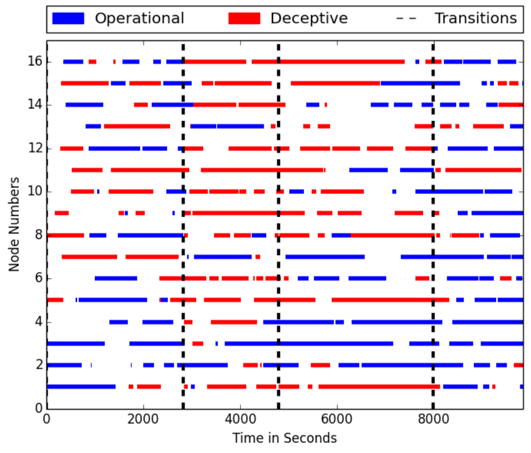
Figure 6.1: Example web-based interface controller to Fortified with 16 nodes

remaining fixed at 1.0. The second transition returns the deceptive rate back to 1.0, and the third transition increases the operational rate to 4.0 while the deceptive rate remains at 1.0. The first and third window of time show equally balanced numbers of operational and deceptive nodes since the maneuver rates are equal. During the second window, we see many more deceptive nodes due to the increased tendency for nodes to maneuver towards the deceptive state versus the operational state. The opposite is true in the fourth window where we see a predominance of operational nodes.

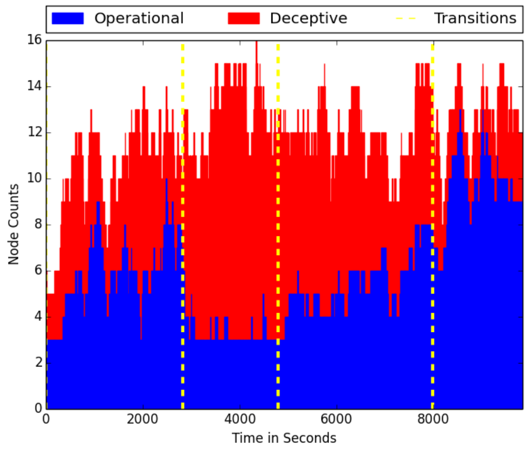
6.2.3 Attacker Modeling

Another optional feature of the Fortified system is the addition of a simulated attacker as described in Chapter 5. The attacker module provides the ability to estimate gains from the moving-target and deceptive defense features of the system. The single attacker is available in both the prototype and simulation modes of execution.

In the discrete event simulation mode of the system, the attacker queries the main controller for a list of the active and deceptive nodes. This retrieval of this list is comparable to the reconnaissance phase that an attacker would undergo to enumerate possible targets and detect potential vulnerabilities in which to exploit. Though our moving-target defenses would have an impact on this phase of an attack, we choose not simulate or study it and reserve it for an area of future study.



(a) Gantt like chart of node maneuvers



(b) Stacked area plot of nodes grouped by nodes

Figure 6.2: Visuals produced by the web-based controller for a 16 node prototype resource maneuver manager

After receiving a list of active nodes, the attacker randomly selects, with uniform probability, a node from the active list. This node is now targeted by the attacker and will remain so until the attacker's compromise timer expires or the node maneuvers. This timer is a random variable drawn from an exponential distribution with rate equal to the attacker compromise rate (λ_{ac}). This rate is configurable within the system and allows comparisons with the operational and deceptive maneuver rates to study the improved security of the system.

The attacker's attempt to compromise the system can have three different outcomes. The first outcome is when the compromise timer expires and the targeted node is an operational node. This is a successful compromise and is advantageous of the attacker but to the disadvantage of the system owner. The second outcome is the compromise timer expires when the targeted node was deceptive. This results in the internal honeypot software of the deceptive node alerting the system owners of the presence of an attacker. This result is to the network security professional advantage because it provides valuable monitoring and intelligence collection opportunities. It is also a disadvantage to the attacker because any continued efforts to exploit the system are under surveillance by the defenders and wasted since no operational capabilities are degraded in the system. The final result is that either of the maneuver timers expires and the attacker is removed from the system. The attacker has wasted time and effort and increases its probability of detection and exposure of exploitation tools. This result has mixed benefits to the system administrator and network defender. When a deceptive node maneuvers to idle and removes the attacker, this is a missed opportunity for the defender to gather information on the attacker.

When using an attacker, the Fortified system running in discrete event simulation mode records a detailed event log of all targeting attempts and result. At the conclusion of the targeting of a node, when maneuver or compromise timer expire, the attacker restarts the targeting process. This action allows us to record various targeting and compromise statistics for analysis of different configurations of the Fortified system.

In the prototype mode of execution the attacker module performs in a similar manner to the simulation mode. However, there are a few exceptions in the behavior. When a node becomes targeted, the attacker launches a process on the remote node. This process sets the compromise timer and waits for it to expire. At expiration, it searches for the operational Hadoop daemons and kills them. It logs this successful defeat and the targeting process restarts. In the event that operational daemons are not detected, it is an indication the targeted node is deceptive. This result is also logged and the targeting process restarts. In the event of a node maneuvering before the compromise timer expires, the maneuvering node searches for and kills the attacker processes. Additional system checks are deployed to insure no previously compromised operational nodes ever rejoin the operational or deceptive clusters. This feature allows us to study the operational capability of a cluster under attack and measure various statistics of a prototype system. This evaluation is a future area of study and is not covered in this chapter.

6.3 Model Verification

Another feature of the Fortified system is the ability to verify the Defense Maneuverable Cyber Platform as described in Chapter 5. Combined with the discrete event simulator and the attacker mode, we add a model verification module to take measurements for the simulation and compare them to our modeled or calculated results. Showing that our discrete event simulation of the system performs as described by the model and various analysis equations in Chapter 5 validates our model.

We evaluate our overall equation (5.8) for calculating the probability of a randomly selected active node not resulting in compromise of an operational node. We refer to this as the probability of surviving an attack. We analyze each component for validation. We strive to maximize each of the components, keeping in mind that the goal is to maximize the overall probability of survival. Once all three individual probabilities are verified, we study the overall probability of survival behavior in simulation and compare it our modeled calculations.

6.3.1 Probability of Defensive Targeting

The probability of defensive targeting is analyzed first. When the attacker chooses a random active node to target, the node can be either deceptive or operational. In the event that a deceptive node is selected, the system survives this attack since it cannot result in an operational node being compromised. In the event that an operational node is selected, other defense features must be utilized to survive this attack. As such, we strive to maximize the probability that a defensive node is targeted.

The calculation of this probability is based on the ratio of operational to deceptive nodes at any given time of the system. By using the steady state probability of the Petri Net model along with the number of operational and deceptive nodes in each marking of the reachability graph, we can compute a weighted average of this ratio of nodes. Recall from previous calculations and observations that solving the steady steady probabilities involves a infinitesimal generator of the size of M^2 where M is the number of markings in the Petri Net. M grows on the order of e_N where N is the number of nodes in the system. Due to the complexity of solving such a large matrix, we use the constrained calculations for smaller networks as shown in Figures 5.4 and 5.5. These calculations are based on networks of size 6, 7, 8 and 9 nodes where the minimum number of operational and deceptive nodes is 3 and 2, respectively.

We run Fortified in discrete event simulation mode for 100 randomize simulations where each simulation consists of 1,000 node maneuvers. We vary the ratio of the operational to the deceptive maneuver rates from (0.25, 1.0, 4.0) and test all four configurations discussed above (6-3-2, 7-3-2, 8-3-2, and 9-3-2). For each simulation, the total number of node-seconds for operational and deceptive nodes is calculated. The probability of a randomly targeted node being deceptive is calculated as the deceptive node-seconds divided by the sum of the operational and deceptive node-seconds. The mean and standard deviation of these percentages is shown in Figure 6.3 along with the calculated probabilities from the Petri-Net solver of PIPE2.

In Figure 6.3, each group of bars represents the same node count configuration and minimum

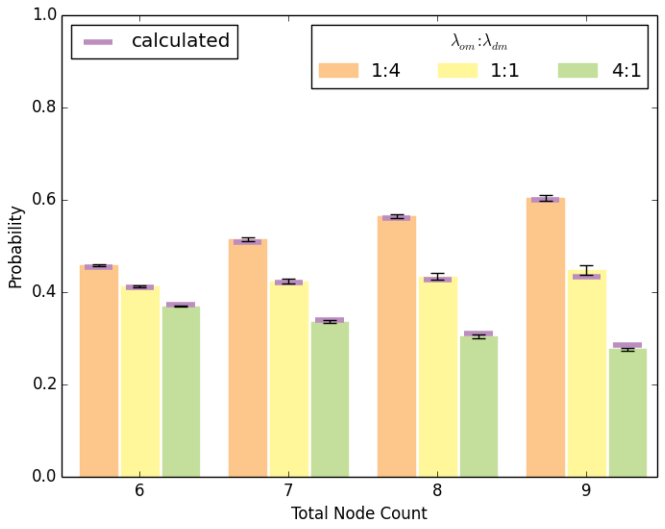


Figure 6.3: Measured vs Calculated Probability of a Random Targeted Node being Deceptive

levels of operational and deceptive nodes. Each individual bar height represents the mean of the probability of a defensive node being targeted. The error bar is one standard deviation from the mean while the horizontal bar represents the calculated probability. Each measured mean is within one standard deviation of the calculated probability of a randomly selected node from all active nodes being deceptive. Figure 6.3 shows that the calculated probabilities the measure are very close for all values tested. These results provide strong support of validation to our model for deceptive node targeting.

6.3.2 Probability of Maneuver Ability

In this section, we study the probability of an operational node being enabled to maneuver to the idle state. Minimum levels of operational nodes are required for some distributed systems to remain operational without loss of data or required operational output. Based on these minimum levels, in some cases operational nodes may be unable to maneuver into the idle state until another node becomes operational. In the minimum and deceptively maneuverable states, operational nodes are static targets and our moving-target defense mechanism provides no additional security. We strive to increase the probability of operational nodes having the ability to maneuver.

When operational and deceptive maneuver rates are equal, we previously have shown this probability is a ratio of the number of fully maneuverable and operationally maneuverable states divided by the total number of markings. When the maneuver rates are not equal, this is calculated as a weighted average where the probability of each marking is taken into consideration. When maneuver rates are equal, all markings are equally probable. The individual markings probabilities are complex calculations as discussed in the previous subsection on the infinitesimal generator of the reachability graph. As such, we only consider systems with equal maneuver rates for the evaluation in this subsection.

In Chapter 5, we present Equation 5.9 to calculate the probability of operational nodes being enabled to maneuver. The model verification feature of Fortified uses this equation to compute the number and compare it to the measurements of when operational nodes are enabled for maneuver. In the simulation, we measure this probability by recording the amount of time in which the number of operational nodes is greater than the minimum and dividing by the total time of the simulation.

Each set of simulations was evaluated by the execution of 100 random experiments each with 10,000 node maneuvers. The percentage of time is calculated for which the operational nodes can maneuver, and the mean and standard deviation is calculated for the 100 experiments. As analyzed in Section 5.4.4, we focus on system configurations where $N = 48$. We run simulations using the statistics observed and recorded in Figure 5.7 where we varied the minimum levels of operational and deceptive nodes. In this previous evaluation, we calculated a "knee" in the curve where the minimum operational node count is fixed and the deceptive node count causes the slope of the probability to exceed the negative diagonal. We choose three different minimum operational node plots to simulate using three specific values of the deceptive node minimum (greater than, less than, and at the knee of the curve). The measured and calculated results are shown in Figure 6.4

Each group of bars represents the same minimum level of operational nodes. Each individual bar height represents the mean of the probability of a operational nodes being enable to maneuver. The error bar is one standard deviation from the mean while the vertical bar represents the calculated probability. All measured means are within one standard deviation of the calculated probability of operational nodes being able to maneuver. These results provide strong support for validation of our model for operational node maneuvering ability.

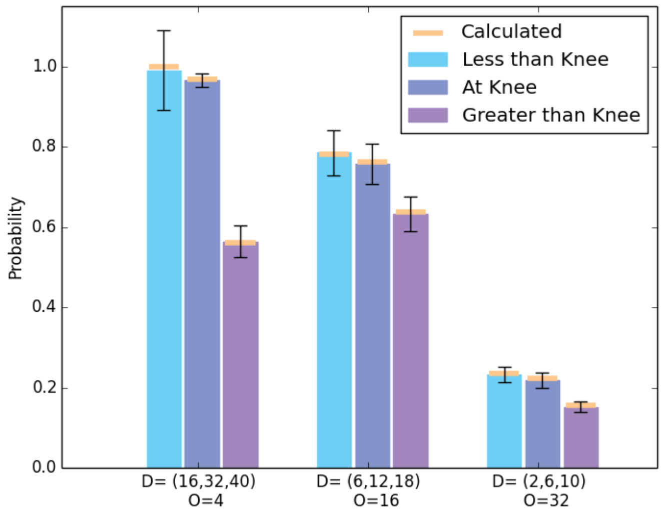


Figure 6.4: Measured vs Calculated Probability of a Operational Node Being Enabled to Maneuver

6.3.3 Probability of Maneuvering Before Attacker Compromise

The final individual probability to validate is the probability of maneuvering before an attacker can compromise the node. When an node is targeted by an adversary, the node survives if the deceptive maneuver timer is less than the attacker compromise timer. We show in Chapter 5 that this probability can be calculated as the probability that one exponential random variable is smaller than another. This probability calculation is a ratio of the first rate divided by the sum of the rates. For our specific application, the probability is calculated as the deceptive maneuver rate λ_{dm} sec:model-prob-maneuver by the sum of the deceptive maneuver rate and the attacker compromise rate $(\lambda_{dm} + \lambda_{ac})$.

To evaluate the correctness of this equation from our model, we run simulations using Fortified to measure the outcome of potential attacks to compare to the model. We conduct these simulations using the optional attacker module of Fortified with various compromise rates equal to 2^n where n ranges between -4 and 4. A static value of 1.0 is used for the two maneuver rates λ_{om} and λ_{dm}. Since the calculation of the probability of the maneuver timer being less than the compromise timer is a proportional calculation based on the ratios of λ_{dm} and λ_{ac}, these setting are sufficient for comparing the calculation and the measurements. A cluster of size $N = 48$ and minimums of $O = 0$, $D = 0$ is used for all simulations. Having no minimum number of operational or deceptive

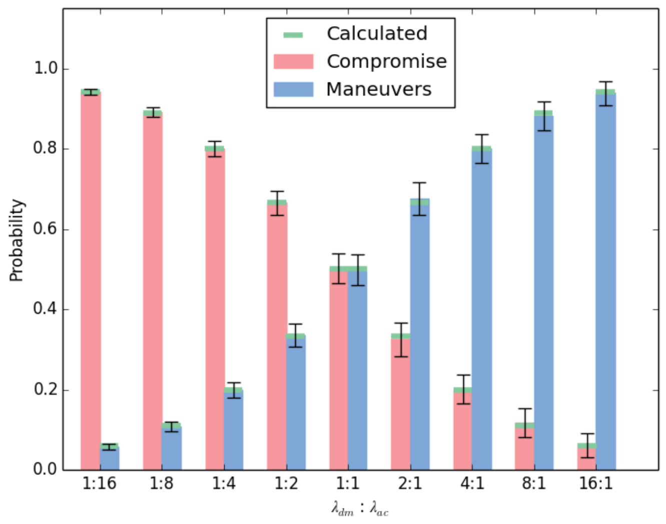


Figure 6.5: Measured vs Calculated Probability of an Operational Node being Compromised or Outmaneuvering an Adversary

nodes guarantees our system remains fully maneuverable throughout the simulation. That is, all compromises of operational nodes are due to the value of the operational maneuver timer and the attacker compromise timer and not impacted by nodes being unable to maneuver.

We simulate each configuration for 100 runs with each run consisting of 1,000 node maneuvers. We record the times an operational node is targeted and the result of the attack as either a maneuver before compromise or an actual compromise. We calculate the percentages for each outcome and computed the mean and standard deviation over the 100 runs. The results are plotted in Figure 6.5 along with the computed probability for each configuration for comparison.

Each group of bars represents a set of simulations where the ratio of the maneuver rate and the compromise rate were equal. The height of the first bar in each group represents the mean probability of an attack resulting in a compromise while the second height represents the probability of a node maneuvering before compromise. The error bar is one standard deviation from the mean while the horizontal bar represents the calculated probability. All measured means fall within one standard deviation of the calculated probability of both outcomes. These results provide strong support for validation of our model for the probability of an operational node's maneuver timer being less than an attacker's compromise timer.

6.3.4 Overall Probability of Survival

After verifying all three probabilities that compose the overall survival probability, the next step is to investigate the combination all three components to validate the entire model. We have shown that the probability of defensive targeting by our modeled attacker is related to the composition of the operational and deceptive nodes in the cluster determined as a function of the two maneuver rates. The probability an operational node is enabled to maneuver has been shown to match the probability the system is fully or operationally maneuverable. We illustrate, without a minimum level of operational and deceptive nodes, the probability of an operational node outmaneuvering an attacker is determined by the deceptive maneuver rate and the attackers compromise rate. Studying these three individual probabilities with multiple configurations in our Fortified system, we validate the overall probability of survival equation (5.8).

To investigate the overall probability of survival, we analyze a cluster composed on nine nodes ($N = 9$) with a minimum level of operational nodes of three ($O = 3$) and the minimum number of deceptive nodes of two ($D = 2$). We refer to this configuration as $9 - 3 - 2$. This configuration has been studied in previous chapters and provides a capstone evaluation for the system. The configuration is large enough to provide interesting results, but at the same time small enough that the PIPE2 Petri Net solver is able to compute individual state probabilities as needed in the calculations.

By applying the previous enumeration metrics (Equations 5.1, 5.4, 5.5, 5.6, 5.7) of the cluster we can show that the 9-3-2 configuration has a total of 9,822 total markings. Of these markings, 1,260 are minimally maneuverable. There are 2,340 operationally maneuverable and 3,528 deceptively maneuverable states. The number of fully maneuverable states is 2,730. A visual of the enumeration of the state space broken down by values of operational and deceptive nodes is shown in Figure 6.6.

Equation 5.3 is used to calculate the total number of edges in the reachability graph for the 9-3-2 configuration. There are 44,604 edges in the graph. Graphically creating such a complex reachability graph with almost 10,000 nodes and over 44,000 edges requires complex processing and would provide little information in this document. But a color coded representation of the infinitesimal generator can provides an indication of the complexity and repetitiveness of the network. Figure 6.7 provides a visual of the sparsely populated matrix of dimensions 9822 x 9822 (96 million

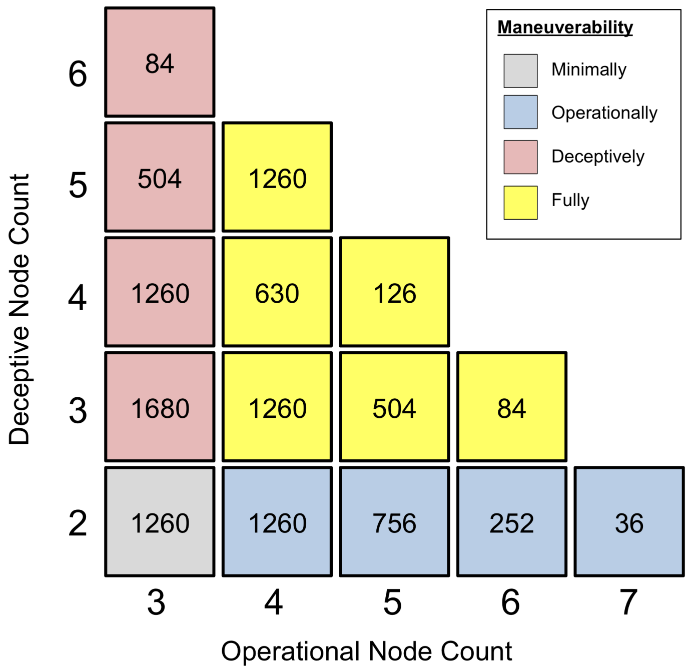


Figure 6.6: Visual depiction of the maneuverable states of a 9-3-2 Fortified System

cells) populated twice by each of the 44,604 edges. The presence of a color represents an edge from the source node on the x-axis to the destination node on the y-axis. Colors are enlarged to be visible and not drawn to scale. A light blue color indicates a transition between an minimally maneuverable state and an operational state, in either direction. A light red spot is an indication of edges between minimally maneuverable and deceptively maneuverable. Light green represent transitions between operationally maneuverable and fully maneuverable. While orange represent transitions between deceptive and fully maneuverable. Dark blue represent edges between operationally maneuverable states, while red represent edges between deceptively maneuverable states. Lastly yellow blocks are transitions from one fully maneuverable state to another.

To validate the correctness of the overall probability model, we ran the Fortified system in a 9-3-2 configuration in discrete event simulation mode over a set of maneuver rates and attacker compromise rates. Three values (1.0, 2.0 and 4.0) are considered for each of the three rates. All 27 possible combinations are explored, though some combinations have similar characteristics since the ratios of the rates and not the actual values impact the behavior of the system. Each system is described as a three-tuple with the operational maneuver rate listed first, followed by the deceptive

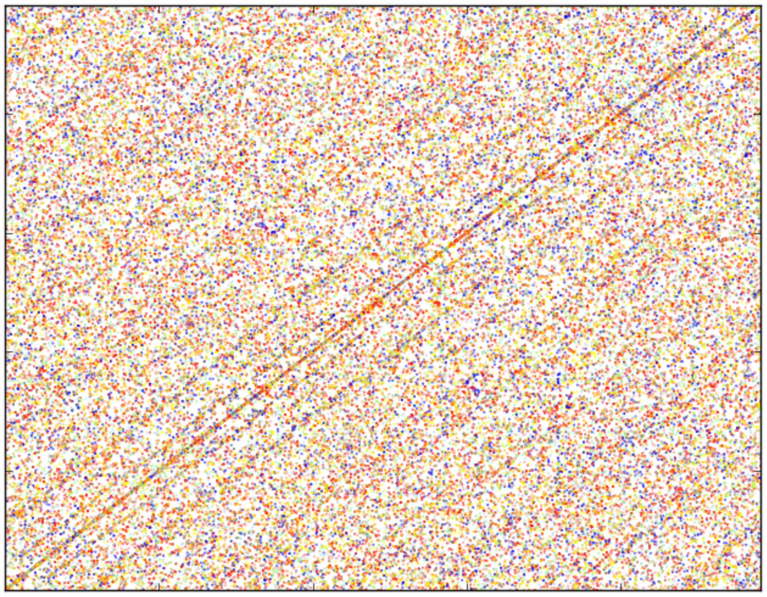


Figure 6.7: Visual depiction of the infinitesimal generator of a 9-3-2 Fortified System

maneuver rate, and concluded by the attacker compromise rate. Each rate is delimited by a dash, so the string 1.0-2.0-4.0 describes a configuration where $\lambda_{om} = 1.0$, $\lambda_{dm} = 2.0$, and $\lambda_{ac} = 4.0$.

Each experiment is executed 100 times for each of the 27 configurations. An experiment consists of running the simulation for a total of 100,000 jumps while the attacker continuously targets an active node. Due to the different values of the maneuver rates and attacker rate, the total count of events when the attacker targeted a node varies for each configuration. The attack count range is approximately between 9,000 and 50,000 with a mean of 22,000. The number of times an operational and deceptive nodes is targeted is recorded along with the two possible results for each type of target (a compromise or an outmaneuver). The percentage of targets resulting in each of the four possible outcomes is calculated. We are interested in the percentage of times in which an attack did not result in an operational node becoming compromised.

Figure 6.8 plots this percentage in the y-axis for each of the 27 configurations as a box-plot of each experiments survival percentage. Each box identifies first and third quartiles, the red line indicates the median, and the whiskers show 1.5 interquartile range (IQR). The results for each configuation is plotted along the same value along the x-axis with the dashed configuration label indicating the values of the maneuver rates and the compromise rate. Additionally, the modeled probability using Equation 5.8 for each configuration is shown as an upward pointing orange triangle.

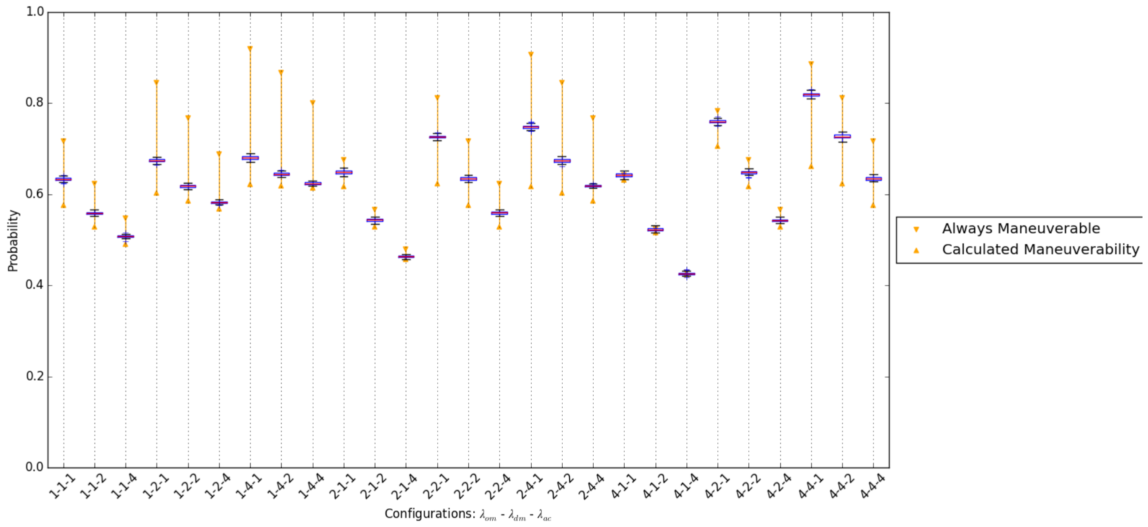


Figure 6.8: Box-plots of survival of 9-3-2 under various rates and calculated bounds

An downward pointing orange triangle is plotted to represent the value of Equation 5.8 at the point where the maximize probability of an operational node is enabled to maneuver, at 1.0. The connecting orange line between the two triangles presents the bounded expected range for the survival probabilities.

Figure 6.8 shows that results based on the model are exceeded by all experiments during all configurations of the system. The probability of survival as shown in Equation 5.8 is a lower bound to the observed results when using the Fortified system in discrete event simulation mode. An upper bound to the plot is equal to the probability of survival when the operational nodes are always able to maneuver $P_m = 1$. For some configurations shown, this range is very large (almost 30%) while for some the range is negligible. The box-plot of all observed survivals from targeting by the attacker always fall within this bound. The observed survival probability typically falls in the lower half of the range when all rates are equal. In some cases when the maneuver rates are 4 times the attacker rate, we see the box-plot fall in the upper half of our range.

Further analysis of the results from the 27 configurations provides understanding of the features that cause the system to behave better than anticipated. We focus on the results of the experiments with regards to each targeting attempt. By dividing the total number of times in which a defensive node is targeted by the total targeting attempts, the probability of defensive targeting

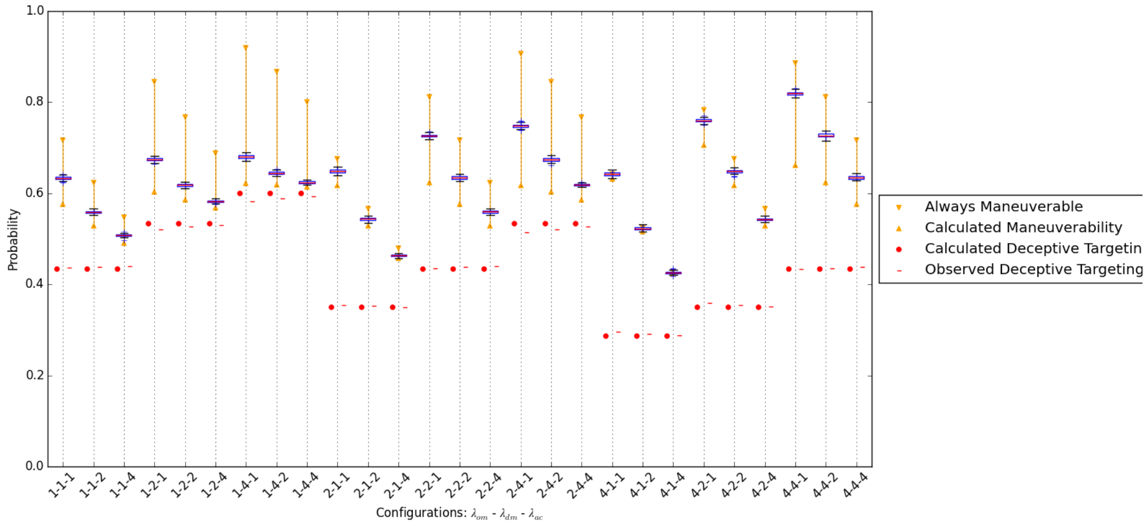


Figure 6.9: Plot of Observed vs Calculated Deceptive Targeting Probability

can be calculated. The means of the observed probabilities of an attacker targeting a deceptive node are plotted alongside their calculated probabilities in Figure 6.9. This plot also includes the previous box-plots of total survivability for comparison. Figure 6.9 shows the observed deceptive targeting probability are very close in proximity to the calculated probability. As the attacker's compromise rate increases, the margins of error decrease. This can be explained because the sample size of the number of attackers increases with a higher attacker compromise rate.

Since deceptive nodes are targeted at the modeled rate, the conclusion can be made that operational nodes are surviving attacks at a higher probabliity than expected. We previously describe a targeted operational node surviving compromise when it is enabled to maneuver and the maneuver time is less than the compromise timer of the attacker. We will refer to probability that an operational node, regardless of maneuverability state, maneuvers before it is compromised as the probability of outmaneuver. The model calculates this probability as the product of the probabilities of maneuver enablement and the probability of maneuver time being less than compromise time. Figure 6.10 shows the calculated outmaneuver probability, along with the observed outmaneuver probability. For comparison, we include the box-plots of the observed survival probabilities of the 27 configurations.

Figure 6.10 shows the observed and calculated outmaneuver probabilities in groups of three.

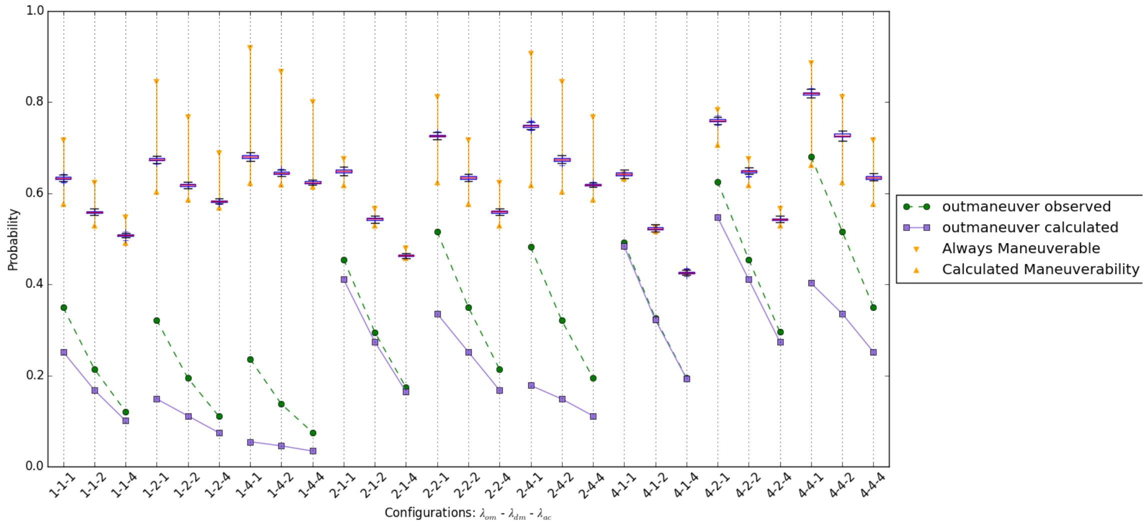


Figure 6.10: Plot of observed vs calculated operational node outmaneuver probability

All three points in each group have the same operational and deceptive maneuver rates. Each point only differs in the attacker's compromise rate. As the compromise rate increases, the observed outmaneuver probability approaches the calculated probability. The explanation of this behavior is due to the decreasing mean for the attacker compromise time as the rate increases. This decreasing window of time cause the sampling of the outmaneuver instances to be a closer match of the steady-state probability of an operational node being enabled to maneuver. When the mean of attacker compromise time is larger, the maneuverability of the operational nodes is better than steady state due to the maneuvers that occur during this window.

Additionally, there are very small bounded ranges for the probability of survival for instances when the operational maneuver rate is more than twice the deceptive maneuver rate. This observation is because as the probability of operational node maneuver enablement approaches 1, the upper and lower bounds approach the probability of the maneuver time being smaller than the compromise time. Thus, there is little difference in the values for upper and lower bounds of the probability of survival.

6.4 Discussion

The model validation results from the discrete event simulation execution mode of the prototype resource maneuver manager allows us to adjust the model. Previously, Equation 5.8 shows the calculation of an absolute value for the probability that a targeted operational node can survive. These results show that that this calculation is actually a worse-case probability. As such, the bounded results for the probably of survival ($P_{survival}$) are presented in Equation 6.1:

$$P_{dt} + (1 - P_{dt}) * P_m * P(t_m < t_c) \leq P_{survive} \leq P_{dt} + (1 - P_{dt}) * P(t_m < t_c) \tag{6.1}$$

Furthermore, we enhance and reinforce the principles provides in Rules of Thumb #3 and #5. The third rule suggests using the ratio of the operational and deceptive maneuver rates to control the operational composition of the cluster. Additionally, it has been shown that increasing the ratio increases the probability of maneuver thus creating a small bounded probability of survival. Though the results in Figure 6.8 also show that a lower ratio between operational and deceptive maneuver rate improves survivability due to its impact on the probability of defensive targeting. The fifth rule of thumb recommends maintaining the deceptive maneuver rate higher than the estimated attacker compromised rate based on threat analysis and intelligence collecting. This recommendation holds true for the ratio of the operational maneuver rate and the attacker compromise rate also. The four best probabilities of survivals shown in Figure 6.8 are when both the operational and deceptive maneuver rates are at least twice that of the attackers rates. The highest probability is when both maneuver rates are four times the compromise rate. This reinforces the importance of system designers to understand not only the capabilities of the attacker but also how fast the distributed system can maneuver between modes of operation.

An updated list of rules of thumb for the defensive cyber maneuverable platform is presented below:

Rule of Thumb #1: Set the minimal operational node parameter just above the operational breaking point of your system.

Rule of Thumb #2: Set the minimal deceptive node parameter as equal to or

just less than the minimum number of operational nodes.

Rule of Thumb #3: Increase the availability of operation nodes in the DMCP by increasing the operational maneuver rate. This also creates a more operationally maneuverable cluster thus increases the ability to outmaneuver an attacker. Maneuver the DMCP to a more deceptive and defendable state by decreasing the operational maneuver rate. Additionally, this action improve operational node survival by increasing the probability a defensive node is targeted.

Rule of Thumb #4: Ensure the operational and deceptive maneuver rates are limited to ensure the underlying distributed and parallel system does not break from having nodes maneuver too frequently.

Rule of Thumb #5: Set the operational and deceptive maneuver rate higher than the estimated attacker compromise rate

In this chapter, we have present research in the area of combining multiple pieces from our preceding systems to build Fortified, the Prototype Resource Maneuver Manager. This software implementation synergistically integrates the resource provisioning aspects of JUMMP along the security enhancements of the DMCP. Fortified will shows how various aspect of military maneuver combined together are used to build prototype maneuver applications will providing enhancements to distributed computing. Additionally, Fortified has verified and enhanced the Petri-Net model for a moving-target and deceptive defense system of a distributed and parallel system

Chapter 7

Conclusion

In this dissertation, we present our motivations for maneuver in cyberspace operations and present our work in designing, building, and modeling maneuverable applications. Our work focuses on provision resources, optimizing applications, and improving cybersecurity. Our work culminates with the integration of our previous works into a prototype resource maneuver manager which validates our model for a maneuver system in the presence of an potential intruder. This work demonstrates how the military concept of maneuver can be applied to distributed and parallel computing in multiple facets.

In the area of resource provisioning, the Job Uninterrupted Maneuverable MapReduce Platform deploys a Hadoop cluster within an existing academic high performance computing environment. JUMMP supports high availability and continuous computing for research and education while incurring no additional financial or administrative overhead. It shows as efficient as a persistent Hadoop cluster on dedicated computing resources, depending on the jump time. The cluster remains stable, with good performance, in the presence of jumps that occur as frequently as the average length of reduce tasks.

As for application optimization, our work with the Flow Optimized Route Configuration Engine provides the design and prototype development of a datacenter testbed with a reprogrammable network topology. Our testbed includes a Virtual Topology Engine that builds virtual network topologies over physical links with SDN flows and a Flow Network Evaluation system to generate a network congestion estimation score. We highlight the design and portray the development of a Hadoop shuffle traffic simulator placing realistic loads on datacenter networks. Experimental results indicate placement of computation racks within a datacenter topology potentially has significant impact of the Hadoop shuffle traffic completion time.

Our research into modeling a Defensive Maneuver Cyber Platform with Stochastic Petri Nets demonstrates cybersecurity improvement through maneuver. This model introduces a distributed and parallel application utilizing moving target defense and deceptive defense tactics to increase survivability in the presence of a cyberattack. We analyze our SPN model to understand the trade-offs between security and operations in defensive maneuver cyber platform. We make recommendations for how using and extending our current SPN model to build prototype systems implementing moving target and deceptive defense in parallel and distributed applications.

Lastly, We present our work in building a prototype maneuver resource manager integrating our previous works. Our prototype system utilizes operational and deceptive Hadoop installations, a discrete event simulator driven by an intelligent scheduler based on modeled parameters for a maneuverable system, and a model verification and validation ability helping to solidify the findings our Petri Net model.

Bibliography

[1] *Amazon Elastic MapReduce (Amazon EMR)*. http://aws.amazon.com/elasticmapreduce/.

[2] Deception and the art of cyber security.

[3] *Deploying MapReduce v2 (YARN) on a Cluster*. http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH4/4.2.1/CDH4-Installation-Guide/cdh4ig_topic_11_4.html.

[4] *HOD Scheduler*. http://hadoop.apache.org/docs/r1.1.2/hod_scheduler.html.

[5] *NetworkX - High-productivity software for complex networks*. http://networkx.github.io/.

[6] *The OpenStack Project*. http://www.openstack.org.

[7] *The Palmetto Cluster*. http://citi.clemson.edu/palmetto/.

[8] *Project Floodlight*. http://www.projectfloodlight.org/.

[9] *Top 500 Supercomputer Site*. http://top500.org.

[10] Faraz Ahmad, Seyong Lee, Mithuna Thottethodi, and T. N. Vijaykumar. PUMA: Purdue MapReduce Benchmarks Suite. *ECE Technical Reports*, October 2012.

[11] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, May 1984.

[12] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74, 2008.

[13] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, volume 10, pages 19–19, 2010.

[14] Ehab Al-Shaer, Qi Duan, and Jafar Haadi Jafarian. Random Host Mutation for Moving Target Defense. In *Security and Privacy in Communication Networks*, pages 310–327. Springer, 2013.

[15] Keith B. Alexander. Warfighting in Cyberspace. *Joint Forces Quarterly*, (46), July 2007.

[16] Scott D. Applegate. The principle of maneuver in cyber operations. In *Cyber Conflict (CYCON), 2012 4th International Conference on*, pages 1–13. IEEE, 2012.

[17] United States Army. *Field Manual 3-0: Operations*. Washington, DC, 2008.

[18] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[19] Theophilus Benson, Aditya Akella, Sambit Sahu, and Anees Shaikh. *Epic: Platform-as-a-service model for cloud networking.* 2011. Published: Technical Report 1686, CS Department, University of Wisconsin, Madison.

[20] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, January 2010.

[21] P. Beraud, A. Cruz, S. Hassell, and S. Meadows. Using cyber maneuver to improve network resiliency. In *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, pages 1121–1126, November 2011.

[22] P. Beraud, A. Cruz, S. Hassell, J. Sandoval, and J.J. Wiley. Cyber defense Network Maneuver Commander. In *2010 IEEE International Carnahan Conference on Security Technology (ICCST)*, pages 112–120, October 2010.

[23] Jonathan Billington, Sren Christensen, Kees Van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber. *The Petri net markup language: concepts, technology, and tools.* Springer, 2003.

[24] CERN. *The Grid: A system of tiers.*

[25] T.M. Chen, J.C. Sanchez-Aarnoutse, and J. Buford. Petri Net Modeling of Cyber-Physical Attacks on Smart Grid. *IEEE Transactions on Smart Grid*, 2(4):741–749, December 2011.

[26] Richard Chirgwin and 26 May 2014. Kiwis unplug supercomputer after intrusion.

[27] G.C. Dalton, R.F. Mills, J.M. Colombi, and R.A. Raines. Analyzing Attack Trees using Generalized Stochastic Petri Nets. In *2006 IEEE Information Assurance Workshop*, pages 116–123, June 2006.

[28] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[29] U.S Department of Defense. *Joint Publication 1-02: Department of Defense Dictionary of Military and Associated Terms.* Joint Chiefs of Staff, Washington, DC, November 2010.

[30] J. Diaz, G. von Laszewski, Fugang Wang, A.J. Younge, and G. Fox. FutureGrid Image Repository: A Generic Catalog and Storage System for Heterogeneous Virtual Machine Images. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 560–564, 2011.

[31] Nicholas J. Dingle, William J. Knottenbelt, and Tamas Suto. PIPE2: a tool for the performance evaluation of generalised stochastic Petri Nets. *SIGMETRICS Perform. Eval. Rev.*, 36(4):34–39, March 2009.

[32] J. Dressler, C.L. Bowen, W. Moody, and J. Koepke. Operational data classes for establishing situational awareness in cyberspace. In *Cyber Conflict (CyCon 2014), 2014 6th International Conference On*, pages 175–186, June 2014.

[33] Matthew Dunlop, Stephen Groat, William Urbanski, Randy Marchany, and Joseph Tront. MT6d: A Moving Target IPv6 Defense. In *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, pages 1321–1326, November 2011.

[34] David Evans, Anh Nguyen-Tuong, and John Knight. Effectiveness of moving target defenses. In *Moving Target Defense*, pages 29–48. Springer, 2011.

[35] David B. Farmer. Do the Principles of War Apply to Cyber War? Technical report, May 2010.

[36] Renato J. O. Figueiredo, Peter A. Dinda, and Jose A. B. Fortes. A case for grid computing on virtual machine. In *Proceedings of the International Conference on Distributed Computing SYstems*, 2003.

[37] Apache Foundation. *Hadoop.* http://hadoop.apache.org/.

[38] Apache Foundation. *Yet Another Resource Negoitator (YARN).*

[39] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[40] M. Hammoud, M.S. Rehman, and M.F. Sakr. Center-of-Gravity Reduce Task Scheduling to Lower MapReduce Network Traffic. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 49–58, 2012.

[41] M. Hammoud and M.F. Sakr. Locality-Aware Reduce Task Scheduling for MapReduce. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 570–576, 2011.

[42] S. Hassell, P. Beraud, A. Cruz, G. Ganga, S. Martin, J. Toennies, P. Vazquez, G. Wright, D. Gomez, F. Pietryka, N. Srivastava, T. Hester, D. Hyde, and B. Mastropietro. Evaluating network cyber resiliency methods using cyber threat, Vulnerability and Defense Modeling and Simulation. In *MILITARY COMMUNICATIONS CONFERENCE, 2012 - MILCOM 2012*, pages 1–6, October 2012.

[43] Brandon Heller, Colin Scott, Nick McKeown, Scott Shenker, Andreas Wundsam, Hongyu Zeng, Sam Whitlock, Vimalkumar Jeyakumar, Nikhil Handigol, James McCauley, Kyriakos Zarifis, and Peyan Kazemian. Leveraging SDN Layering to Systematically Troubleshoot Networks. In *Proceedings of ACM Workshop on Hot Topics in Software Defined Network*, 2013.

[44] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. ElasticTree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[45] Robert L. Henderson. Job scheduling under the Portable Batch System. In DrorG. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 279–294. Springer Berlin Heidelberg, 1995.

[46] Marius Hillenbrand. *Towards Virtual InfiniBand Clusters with Network and Performance Isolation.* System Architecture Group, Karlsruhe Institute of Technology (KIT), Germany, 2011.

[47] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked System Design and Implementation*, 2011.

[48] Hortonworks. *Best Practices for Selecting Apache Hadoop Hardware.* http://hortonworks.com/blog/best-practices-for-selecting-apache-hadoop-hardware/, 2011.

[49] S. Ibrahim, Hai Jin, Lu Lu, Song Wu, Bingsheng He, and Li Qi. LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 17–24, 2010.

[50] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 127–132, New York, NY, USA, 2012. ACM.

[51] Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, Cliff Wang, and X. Sean Wang. *Moving Target Defense.* Springer, 2011.

[52] Kurt Jensen. Coloured Petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, number 254 in Lecture Notes in Computer Science, pages 248–299. Springer Berlin Heidelberg, January 1987.

[53] Jan Kallberg and Bhavani Thuraisingham. Cyber OperationsBridging from Concept to Cyber Superiority. *Joint Forces Quarterly*, 68, 2013.

[54] Yunhee Kang and Geoffrey C. Fox. Performance Evaluation of MapReduce Applications on Cloud Computing Environment, FutureGrid. In Tai-hoon Kim, Hojjat Adeli, Hyun-seob Cho, Osvaldo Gervasi, Stephen S. Yau, Byeong-Ho Kang, and Javier Garca Villalba, editors, *Grid and Distributed Computing*, number 261 in Communications in Computer and Information Science, pages 77–86. Springer Berlin Heidelberg, January 2011.

[55] Sriram Krishnan, Mahidhar Tatineni, and Chaitanya Baru. myHadoop-Hadoop-on-Demand on Traditional HPC Resources. In *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego*, 2011.

[56] Jae Woo Lee, Roberto Francescangeli, Jan Janak, Suman Srinivasan, Salman A. Baset, Henning Schulzrinne, Zoran Despotovic, and Wolfgang Kellerer. *NetServ: Active Networking 2.0.* 2011.

[57] Zhao Li, Yao Shen, Bin Yao, and Minyi Guo. OFScheduler: A Dynamic Network Optimizer for MapReduce in Heterogeneous Cluster. *International Journal of Parallel Programming*, pages 1–17.

[58] S. Liles, M. Rogers, J.E. Dietz, and D. Larson. Applying traditional military principles to cyber warfare. In *2012 4th International Conference on Cyber Conflict (CYCON)*, pages 1–12, June 2012.

[59] William J. Lynn. Defending a New Domain: The Pentagon's Cyberstrategy. *Foreign Affairs*, pages 97–108, 2010.

[60] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[61] J. Yates Monteith, John D. McGregor, and John E. Ingram. Hadoop and its evolving ecosystem. In *Proceedings of the 5th International Workshop on Software Ecosystems, hosted by 4th International Conference on Software Business (ICSOB 2013)*, 2013.

[62] W.C. Moody, Hongxin Hu, and A. Apon. Defensive maneuver cyber platform modeling with Stochastic Petri Nets. In *2014 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 531–538, October 2014.

[63] William Clay Moody, Jason Anderson, Kuang-Ching Wang, and Amy Apon. Reconfigurable Network Testbed for Evaluation of Datacenter Topologies. In *Proceedings of the Sixth International Workshop on Data Intensive Distributed Computing*, DIDC '14, pages 11–20, New York, NY, USA, 2014. ACM.

[64] William Clay Moody, Hongxin Hu, and Amy Apon. Modeling Moving Target and Deceptive Defenses of Distributed and Parallel Systems with Stochastic Petri Nets. *EAI Endorsed Transactions on Collaborative Computing (in review)*, 2015.

[65] William Clay Moody, Linh Bao Ngo, Edward Duffy, and Amy Apon. JUMMP: Job Uninterrupted Maneuverable MapReduce Platform. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8, September 2013.

[66] Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, and Jeffrey C. Mogul. SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies. In *NSDI*, pages 265–280, 2010.

[67] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[68] S. Narayan, S. Bailey, and A. Daga. Hadoop Acceleration in an OpenFlow-Based Cluster. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 535–538, 2012.

[69] Juniper Networks. *Understanding ECMP Flow-Based Forwarding.* http://www.juniper.net/techpubs/en_US/junos13.3/topics/concept/routing-policy-security-ecmp-flow-based-forwarding-understanding.html.

[70] L. Nixon. The Stakkato Intrusions: What Happened and What Have We Learned? In *Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06*, volume 2, pages 27–27, May 2006.

[71] Hamed Okhravi, Adam Comella, Eric Robinson, and Joshua Haines. Creating a cyber moving target for critical infrastructure applications using platform diversity. *International Journal of Critical Infrastructure Protection*, 5(1):30–39, March 2012.

[72] R.C. Parks and D.P. Duggan. Principles of Cyberwarfare. *IEEE Security Privacy*, 9(5):30–35, September 2011.

[73] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 165–178, New York, NY, USA, 2009. ACM.

[74] James L. Peterson. Petri Nets. *ACM Comput. Surv.*, 9(3):223–252, September 1977.

[75] C. Ramchandani. ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY TIMED PETRI NETS. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

[76] D Raymond, G Conti, T Cross, and M Nowatkowski. Key Terrain in Cyberspace: Seeking the High Ground. In *Cyber Conflict (CyCon), 2014 6th International Conference on*, June 2014.

[77] A. Rosen and K.-C. Wang. Steroid OpenFlow Service: Seamless Network Service Delivery in Software Defined Networks. In *Proceedings of the First GENI Research and Educational Experiment Workshop*, 2012.

[78] Robert B. Ross and Rajeev Thakur. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 391–430, 2000.

[79] Sheldon M. Ross. *Introduction to Probability Models, Ninth Edition.* Academic Press, Inc., Orlando, FL, USA, 2006.

[80] Jeff Rowe, Karl N. Levitt, Tufan Demir, and Robert Erbacher. Artificial Diversity as Maneuvers in a Control Theoretic Moving Target Defense. In *National Symposium on Moving Target Research*, 2012.

[81] Zoe Sebepou, Kostas Magoutis, Manolis Marazakis, and Angelos Bilas. A Comparative Experimental Study of Parallel File Systems for Large-Scale Data Processing. In *Proceedings of the 1st USENIX Workshop of Large-Scale Computing*, volume 8, pages 1–10, 2008.

[82] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. *Flowvisor: A network virtualization layer.* 2009. Published: OpenFlow Switch Consortium Technical Report.

[83] P. K. Singh. Maneuver Warfare in Cyberspace. Technical report, DTIC Document, 1997.

[84] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[85] Sayantan Sur, Hao Wang, Jian Huang, Xiangyong Ouyang, and Dhabaleswar K. Panda. Can High-Performance Interconnects Benefit Hadoop Distributed File System. In *Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction with MICRO*, 2010.

[86] Chee-Wooi Ten, Chen-Ching Liu, and G. Manimaran. Vulnerability Assessment of Cybersecurity for SCADA Systems. *IEEE Transactions on Power Systems*, 23(4):1836–1846, November 2008.

[87] W. Tirenin and D. Faatz. A concept for strategic cyber defense. In *IEEE Military Communications Conference Proceedings, 1999. MILCOM 1999*, volume 1, pages 458–463 vol.1, 1999.

[88] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM.

[89] Guohui Wang, T.S. Eugene Ng, and Anees Shaikh. Programming your network at run-time for big data applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 103–108, New York, NY, USA, 2012. ACM.

[90] Richard Wang, Dana Butnariu, and Jennifer Rexford. OpenFlow-based server load balancing gone wild. In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[91] Yandong Wang, Xinyu Que, Weikuan Yu, Dror Goldenberg, and Dhiraj Sehgal. Hadoop acceleration through network levitated merge. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 57, 2011.

[92] Kevin C. Webb, Alex C. Snoeren, and Kenneth Yocum. Topology switching for data center networks. In *Hot-ICE Workshop*, 2011.

[93] Justin Yackoski, Harry Bullen, Xiang Yu, and Jason Li. Applying Self-Shielding Dynamics to the Network Architecture. In *Moving Target Defense II*, pages 97–115. Springer, 2013.

[94] Rui Zhuang, Su Zhang, Scott A DeLoach, Xinming Ou, and Anoop Singhal. Simulation-based approaches to studying effectiveness of moving-target network defense. In *National Symposium on Moving Target Research*, 2012.