Autonomic Model for Self-Healing and Self-Protection in Grid Computing using Multi-Agents

A thesis submitted
for the award of the degree of
DOCTOR OF PHILOSOPHY

by
Inderpreet Chopra
(90703501)

under the guidance of

Dr. Maninder Singh
Associate Professor
Computer Science and Engineering Department
Thapar University, Patiala -147004

Computer Science and Engineering Department
Thapar University, Patiala – 147004, INDIA

April, 2015

Dedicated to God Almighty

For all his grace, mercy and strength that has sustained me throughout this time of my life.

Certificate

I hereby certify that the work which is being presented in this thesis entitled *Autonomic Model for Self-Healing and Self-Protection in Grid Computing using Multi-Agents*, for the award of degree of Doctor of Philosophy submitted to the Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Maninder Singh and refers other researchers works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(Inderpreet Chopra)
Reg. No. 90703501

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Maninder Singh)
Associate Professor
Department of Computer Science & Engineering
Thapar University, Patiala 147 004, Punjab, INDIA

Acknowledgements

PhD is a long and amazing journey all about learning. Many people appear in the path of this journey who enrich your knowledge and life. To all of them, I would like to extend my most heartfelt thanks:

My sincerest gratitude to my supervisor Dr. Maninder Singh, Associate Professor, Computer Science and Engineering Department, for his immense help, guidance, stimulating suggestions and encouragement all the time with this thesis work. He always provide me motivating and enthusiastic atmosphere to work with, it was a great pleasure to do this thesis under his supervision.

I also take the opportunity to thank Dr. Deepak Garg, Associate Professor and Head, Computer Science and Engineering Department, Thapar University, Patiala, for providing us with the adequate infrastructure in carrying the research work.

I would like to express my sincere thanks to Doctoral Committee Members Dr. R.K. Sharma and Dr. Rajesh Kumar for their critical observations and valuable comments which helped enormously in presenting results and shaping this thesis.

I would like to thank faculty and staff members of Computer Science and Engineering Department of Thapar University, Patiala for providing and sharing resources that have been utilized in implementation of

SHAPE. I would like to express special thanks to Dr. Anju Sharma and Dr. Inderveer Chana for always there to answer my all stupid queries.

I am also grateful to my colleagues and management at Expicient Software Pvt. Ltd, Gurgaon for their all cooperation. They always helped me to spare time for research.

This research has profited from the friendship, advice, encouragement and support of several remarkable people. Ms. Ratinder Kaur without any doubt, top of this list for her continual counsel, willingness to listen, considered insights and enthusiasm. Dr. Rajni Aron, Dr. Shashi Bhanwar, Mr. Amit Kumar Bharadwaj and Ms. Jyotsana Sharma whose experienced guidance helped me to overcome my fears and work hard to complete my thesis.

I am also grateful to all my friends and numerous others who have directly or indirectly contributed towards carrying out the research in all aspects.

Finally my heartiest gratitude goes to my family. Without your constant encouragement, un-conditional love and support I would not been able to undertake this endeavour.

Above all, I wish to thank the Almighty God, who heard me when I called him.

Inderpreet Chopra

April 2015

Abstract

Grid Computing has evolved as the new level of distributed system by combining large number of dynamic heterogeneous resources to work as single unit. This helps in utilizing enormous power of large number of computational resources to solve highly computational specific problems. However, the complex nature of grid computing, also brings with it the increase in rate of failures and security attacks which are difficult to handle manually. In this thesis, an automated model for self-healing and self-protection of grid environment using multi-agents is presented to deal with such failures and security attacks.

The major contribution of the thesis is a model called SHAPE. SHAPE stands for self healing and protection environment. Self-healing and self-protection properties of autonomic computing are used to provides a holistic approach for the design and development of SHAPE. This helps grid environment to adapt itself to meet the requirements of fault tolerance and security from attacks without manual intervention.

SHAPE is novel idea that provides many features: (a) It is the first initiative that provides the capabilities for both Self-Healing and Self-Protection in one system. (b) In terms of fault handling, it uses both active and proactive approaches to provide the functionalities to

deal with hardware, software and network failures for distributed systems. (c) For self-protecting system from attacks, it has intelligence to keep updating the network profile of intrusion detection system (IDS) to provide protection from attacks like Distributed denial of Service (DDoS), Remote to Local (R2L), User to Root (U2R), and Probing attacks. (d) From agent-based component design principle, SHAPE is highly scalable, robust and reliable model.

Architecture of SHAPE is based on multi-agent component architecture which consists of two broad categories namely self-healing and self-protection. Under self healing, different algorithms are proposed for monitoring and handling hardware, network and software failures. For implementing monitors for failure handling, Q-learning based approach has been extended to work in distributed environment with minimal overhead. Another unique feature in self-healing category is hardware driver hardening to reduce the failure rate because of hardware failures. Under self-protection, support vector machine (SVM) has been used to provide intelligence into the grid environment to handle various security attacks. This provides dynamic intrusion detection system which keep on updating based upon attack activity profiling.

SHAPE is validated on Grid environment setup using Globus 4.0 within Thapar University campus. Validations are done based upon standard metrics for fault handling and security. Results are published to research community in form of peer-reviewed journal publications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since last two decades there has been a significant increase in commodity computer and network performance due to faster hardware and more sophisticated software. However, there are still some problems related to space, business, science and engineering domains requiring huge processing efforts that cannot be achieved with the current generation of supercomputers. In fact, due to the complex nature of these problems it is not possible to perform compute intensive and data intensive tasks with a single machine. Many experimental studies have been conducted on the cooperative use of geographically distributed heterogeneous resources conceptualized as a single powerful computer. This new approach is well-known by several names, such as, meta-computing, scalable computing, global computing, grid computing and more recent cloud computing.

This chapter provides a high level overview of grid computing, fault tolerance in grid, grid security and role of agents in enhancing grid capabilities. Chapter briefly discusses the need of automating the fault-tolerance and security handling processes in grid middleware. Chapter ends with a discussion on organization of

the rest of the thesis.

1.1 Grid computing

Numerous ideas behind grid computing have been around in various forms throughout the history of computing [79]. Ian Foster coined the term "grid" in mid 1990s to describe a large scale distributed computing infrastructure for advanced science and engineering. Since then, it has emerged as an important and interesting field that is different from conventional distributed computing. It focuses on large scale resource sharing, innovative applications, and, in some cases, high performance orientation [60][61].

The grid infrastructure and paradigm is analogous to electric power grid [99]. Grid supports seamless virtualized resources in order to provide access to effectively infinite computing cycles and data storage for the registered user. All the employed grid resources like infrastructure managing machines, networks, storages, etc. are largely hidden from the user, in the same way, as individuals have no idea which power company, transformer and generator is being used when they plug their electric appliance into the socket. Figure 1.1 shows the sample Grid environment in which application scientists and Grid users from different sites or domains share various types of distributed resources all over the world [105]. Grids are mostly engaged in executing applications that are complex and are highly computational or data intensive.

The Open Grid Services Architecture (OGSA) [59] is the first effort to standardize grid functionality, based on concepts from the web services community. The objectives of OGSA [76] are: to provide a common base for autonomic man-

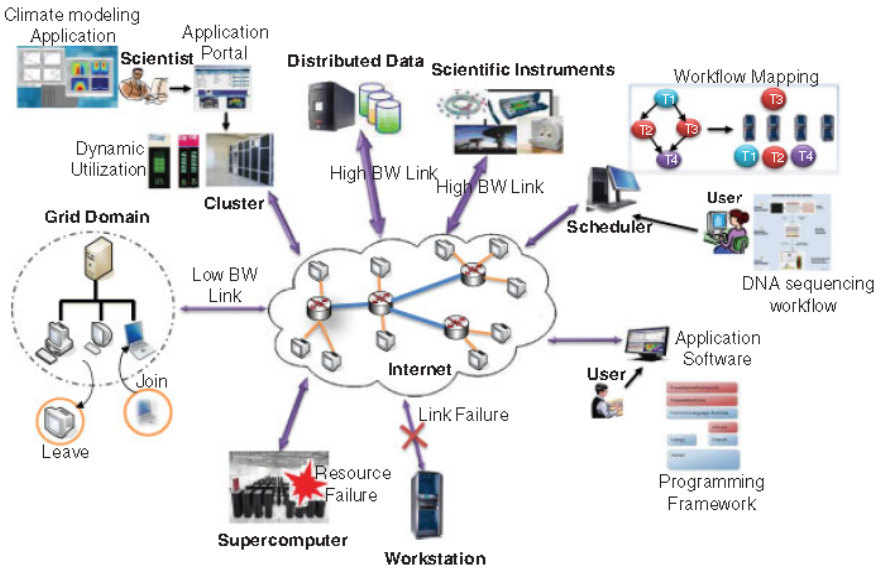


Figure 1.1: Grid heterogenous and dynamic environment[105]

agement solutions to manage heterogeneous resources when necessary; to define open and published interfaces for resource interoperability; to manage heterogeneous resources across heterogeneous platforms; to meet with QoS requirements and to exploit integration technologies in place.

Grid systems are categorized as computational, data and service grids [85] [110]. Data grid is an integrated view of data storage [6]. Every machine connected to the grid provides some quantity of storage for grid use. Storage can be primary memory or secondary memory using hard-disk or some other type of permanent storage media like magnetic disc or tape, etc. The computational grid category denotes the systems that have a higher aggregate computational capacity available for single applications than the capacity of any constituent machine in the system. The service grid category provides specific services that are not provided by any other machine in grid. This category is further divided into on demand, collaborative, and multimedia grid systems. Here in our research,

emphasis is on computational grids.

1.2 Computational grid

Computational grid is a collection of distributed and heterogeneous resources that can be used as an ensemble to execute large-scale applications [76]. This infrastructure provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

The above definition includes some of the specific words having specific meaning concerned to that definition. **Dependable** service means assurance to the user that they will receive predictable, sustained, and often high levels of performance from the diverse components that constitute the Grid. In the absence of such assurances, applications will not be written or used. The need for **consistency** of service is a second fundamental concern. It assures the use of standard services, protocols and interfaces to hide heterogeneity of resources while allowing scalability. Without such standards, application development and pervasive use are impractical. As with electric power, we need standard services accessible via standard interfaces and operating within accepted parameters. **Pervasive** access allows the user to count on services always being available, within whatever environment we expect to move. Pervasiveness does not imply that resources are everywhere or are universally accessible but that the grid will extract maximum performance from the available resources. Finally, this infrastructure offers **inexpensive** (relative to income) access if it is to be broadly accepted and used. Home users and industrialists both make use of remote billion-dollar power plants on a daily basis because the cost to them is reasonable.

Foster et al. have recognized three approaches to build computational grids. These are: the commodity approach, the service approach, and the integrated architecture approach [57]. In the commodity approach, existing commodity technologies, e.g., HTTP, CORBA, COM, Java, serve as the basic building blocks of the grid. In the service approach, as demonstrated by the Globus project, a set of basic services such as security, communication, and process management are provided and exported to developers in the form of a toolkit. In the integrated architecture approach, resources are accessed through a uniform model of abstraction. Since the Grid resources belong to different administrative domains and are geographically distributed, their availabilities may be very dynamic.

1.3 Grid computing to Cloud computing

Cloud computing evolves from grid computing and relies on its backbone and infrastructure support[124][97]. They share a common vision of providing computing as a utility to consumers. The transformation from grid to cloud has been the result of a shift in focus from an infrastructure that delivers storage and compute resources (such is the case in grids) to one that is, an economy based aiming to deliver more abstract resources and services (such is the case in clouds)[63].

Cloud computing enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources like networks, servers, storage, applications, and services, that can be rapidly provisioned and released with minimal management effort or service provider interaction [140]. Both grid and cloud computing share mostly same characteristics with few exceptions[113][89].

- *Heterogeneity*: Grids and clouds involve a multiplicity of resources that

are heterogeneous in nature and might span a number of administrative domains across wide geographical distances.

- *Adaptability and Scalability*: Both grid and cloud adapt themselves to the changing user needs. However, the distinguishing characteristic of cloud is scalability as per the user demand. Resources may scale up equally well as they scale down with changing user needs.

- *Resource sharing*: Grids and clouds are based on virtualization to enable resource sharing. Heterogeneous resources located across multiple administrative domains are pooled to fulfill the resource needs of user applications. One difference is that Grids rely on batch systems for job execution whereas in clouds the resource management is done based upon utilization of virtualization technologies [123].

- *Security*: Both grid and cloud computing allow users to access a shared infrastructure. Users must be authenticated and authorized to maintain the confidentiality and integrity of data and shared resources.

1.4 Need for autonomic model

In order to achieve above characteristics by a single system is a tedious task. Self management of resources is needed to keep the system working at full pace most of time. Self-management can be achieved in three steps [111]. In the first step, information infrastructure is required to provide sufficient information for system awareness. Then awareness triggers decisions which are deduced using system knowledge. In the last step, the decisions taken are executed by exploiting

the adaptive capabilities of the system. In general, the Grid aims to be self-configuring, self-tuning and self-healing, similar to the goals of an autonomic computing. Autonomic computing provides an approach by enabling the design and development of systems or applications that can adapt themselves to meet the requirements of performance, fault tolerance, reliability, security, etc., without any manual intervention.

1.4.1 Autonomic computing

The main objective of Autonomic Computing is to reduce the management complexity of large computing systems. It solves this problem through a smart and increased automation thus exempting the system administrators from many burdensome activities [8]. The basic approach of autonomic computing is to organize the management of functionalities, efficiencies and the quality of services in large computing systems through logically distributed and autonomous controlling elements [104]. The purpose of this is to achieve an amicable functioning of the global system within the confines of its designated behavior while the individual elements take local autonomous decisions. In this approach, there is a shift from a resource entitlement model to a goal-oriented model. In order to significantly reduce system management complexity, one must clearly define the boundaries of these controlling elements. The complexity of controlling elements is mainly reduced by making significant amount of local decisions in these elements. If the local decision process is associated with a smaller time constant, it is easy to revise the decision process and to avoid large damage globally.

Autonomic computing is a perception for attempting self-management in

the system with respect to four areas, which includes self-configuration, self-protection, self-healing and self-optimization. These areas are under constant investigation by large number of researchers and administrators. Thus autonomic computing is an attempt to consolidate related research on areas of computer self-management.

1.4.1.1 Autonomic computing architecture

The architecture of autonomic computing [64] formalizes a reference framework that identifies common functions required to achieve autonomous behavior. Following are the building blocks of autonomic computing:

- *Task manager*: It enables IT personnel to perform management functions through a consistent user interface.
- *Autonomic manager*: It automates common functions and management activities using an autonomic control loop [Figure 1.2]. Through this control loop, autonomic managers monitor resource details, analyze those details, plan adjustments, and execute the planned adjustments-using both information from humans (administrators), as well as rules and policies, both defined (by humans) and learned by the system.
- *Knowledge source*: It provides information about the managed resources and data required to manage them, such as business and IT policies.
- *Enterprise service*: It leverages web standards to drive communications among components throughout the environment.

- *Touchpoint*: It provides a standardized interface for managed resources-servers, databases, storage devices, etc. enabling autonomic managers to sense and effect behavior within these resources.

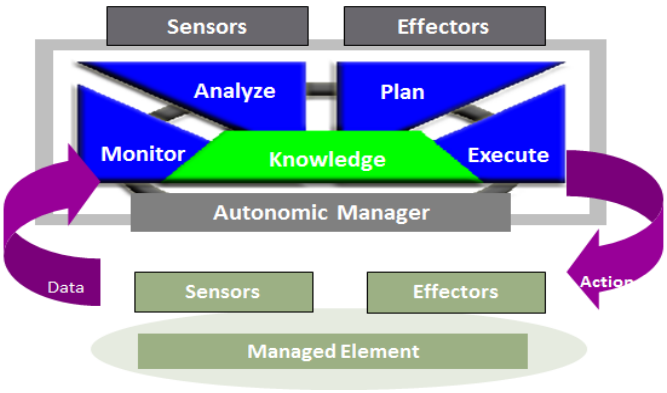


Figure 1.2: Autonomic control loop[64]

1.4.1.2 Features of autonomic systems

The following are the major features that should be incorporated in any system to be classified as an autonomic system. [112].

- *Self-protection*: Self-protection enables the system to secure itself from various types of network attacks. It proactively detects malicious activities and triggers countermeasures in order to stop them. It also helps to overcome the drawbacks of manual management of the system which may results in slow speed, increase in errors and unmanageability.

- *Self-optimizing*: Self-optimizing enables system components to dynamically tune themselves to meet end-user or business requirements with minimum human intervention. The tuning action involves the automatic controlling

and reallocation of resources based upon load balancing functions and system run-time information. This improves the overall resource utilization and system performance in the system.

- *Self-healing*: Self-healing is the ability of a system to recover itself from faults that may cause malfunctioning in system components. A self-healing system must be able to recover from a failed component by first detecting and isolating the failed component, taking it off-line, fixing and reintroducing the fixed or replacement component into service without any apparent overall disruption. It must also predict problems and take actions to prevent the failure from having an impact on applications. The self-healing objective must be to minimize all outages in order to keep the system up and available at all times.

- *Self-configuring*: A self-configuring system must adapt automatically to the dynamically changing environments in such a way that the software or hardware components can be added at run-time with no disruption to other system services with minimal human interaction.

- *Self-learning*: For self-learning the system must be integrated with some machine learning components to build knowledge rules. These rules will become more stable with time and this will in turn improve the system performance, robustness and resilience and anticipation of foreseen failures.

Amongest these features, this thesis work highlights self-healing and self-protection in grid systems.

1.5 Research motivation

Grid computing enables aggregation and sharing of geographically distributed computational, data and other resources as single, unified resource for solving large-scale compute and data intensive computing application. Management of resources in grid computing environment becomes complex as the resources are geographically distributed and heterogeneous in nature. Due to this complex nature of grid, number of failures and security breaches increase exponentially. Therefore, it is of paramount importance to design a mechanism for checking and handling faults causing failures efficiently in the grid infrastructure. Failures are different from faults. A fault is condition that causes the system to fail to perform its required function, whereas failure is the inability of a system or component to perform its required functions within specified performance requirements.Faults may or may not lead to a failure, depending upon the sequence of them happening or the measures taken to treat them.

1.5.1 Self-healing

A self-healing system is able to heal itself at runtime in response to changing environmental or operational circumstances [71]. It has the capability to modify its actions in response to changes such as system faults and resource variability. Need for such systems is felt as coordination among various resources in a distributed environment have become complex due to the heterogeneous nature of computing, data and network resources [152]. Failure detection methods developed for current distributed systems are not regarded suitable for large, heterogeneous and dynamic grid systems. Various questions that are of concern are:

- What are the most frequent types of failures faced on complex distributed systems?
- What are the mechanisms used for detecting and/or correcting and/or tolerating faults?
- What are the major problems encountered while recovering from a failure scenario?
- To what degree is the user involved during the failure recovery process?
- How to re-allocate task and resource automatically in case of failures?
- How can the historic fault information available for node be helpful in increasing throughput?

1.5.2 Self-protection

A self-protecting system can proactively detect and identify hostile behavior and can take autonomous actions to defend itself against malicious or intrusive behavior. Self-protecting systems, as anticipated, could safeguard themselves against two types of behavior: accidental human errors and malicious intentional actions. To protect themselves against accidental human errors, self-protecting systems could provide a warning if the system administrators were to initiate a process that might interrupt services. To defend themselves against malicious intentional actions, self- protecting systems would scan for suspicious activities and react accordingly without users being aware that such protection is in the process. Various questions that are of concern are:

- How to generate signatures for bad traffic?
- What is strategy for reducing false positives?
- How to handle new attacks on the system?
- How the scaling of the security happens in heterogeneous environments like grid?

1.5.3 Multi-agent systems and Grid computing

A Multi-agent system (MAS) is defined as the collection of autonomous agents that work together to achieve common task [81][21]. MAS provides following characteristics:

- *Self-determination*: Agents have control over their actions and can work without the direct human intervention.
- *Social interaction*: Agents interact with other fellow agents to perform the tasks associated with them.
- *Susceptibility*: agents discover their environment and respond in a timely fashion to all the changes happening in the environment.
- *Pro-activity*: Agents has intelligence to take self-initiative and respond to all environment changes. An agent is capable of handling complex, high-level tasks. For this agents might divide complex tasks in small sub-tasks to reduce their complexity.
- *Movable and Flexible*: Agents have ability to transfer themselves to other machines and still retain their current state.

As discussed already, managing grid resources manually is not easy job, as grids involve large-scale heterogeneous resource sharing and high performance computing. Based upon the above characteristics, MAS based approach helps in solving grid challenges by providing:

- *Scalability*: As grid size increases, the need for scalability of the environment increases. This can be achieved by decentralizing grid services. Agents are the best option for implementing decentralized services.

- *Adaptability*: Availability of large number of heterogeneous resources participating in formation of grid keep on changing. Grid system must learn to adapt itself to this changes. This needs autonomous entities. Using MAS, system is forced to perform independent monitoring tasks to look for abnormal behavior.

- *Reliability*: By reliability, it means that the system should be able to tolerate failures and attacks and recover from them. This needs robust entities.

- *Manageability*: Managing grids includes various areas like performance analysis, complexity, fault tolerance and resource management. This can be achieved through intelligent entities working together but performing different kind of managements.

1.6 Key Contributions

Following are the key contributions:

- An important problem of automation of distributed system management in

aspect of recovery from different kind of faults, as well as from number of security attacks were discussed.

- A novel approach called SHAPE is presented that provides the platform for adding self-healing and self-protection capabilities to any distributed system. It is designed using component-based architecture in which one can easily add or remove components.

- To add intelligence into the system, machine learning algorithms using Support Vector Machine (SVM) and reinforcement learning (RL) are used. This helps to automatically handle failures and security attacks based upon the gathered historical data related to failures and security attacks.

- SHAPE has built in capability to handle network, software, and hardware related faults. It also hardens the system so as to reduce the frequency of failure occurrence.

- SHAPE provides a feature for protection against four kinds of security attacks- (Distributed denial of Service) DDoS, (Remote to Local) R2L, (User to Root) U2R, and Probing.

- SHAPE has been evaluated using standard metrics for failures and security attacks in a grid environment. This includes throughput, turn around time, waiting time, detection rate and false positive rate based validations. Results show that SHAPE increases the job execution rates by reducing the security attacks and failures in the system.

1.7 Thesis outline

This section discusses the framework of this thesis. Thesis is organized as follows:

Chapter 1 *Introduction*

This chapter discusses the evolution of grid computing and its usage today as a powerful distributed computing paradigm. It provides a high level overview of grid computing, fault tolerance in grid, grid security and the role of agents in enhancing grid capabilities. Chapter briefly discusses the need of automating the fault-tolerance and security handling processes in grid middleware. Chapter ends with a discussion of the organization of the rest of the thesis.

Chapter 2 *Literature Review*

Chapter 2 reviews and analyzes need for self-healing and self-protection in heterogeneous and dynamic environment such as Computational Grids. It summarizes the kind of faults and security attacks that can occur in grids and various ways to deal with these. Also, a survey describing the various self-healing and self-protection approaches is presented. The chapter finally concludes with the problem formulation and objectives.

Chapter 3 *SHAPE: Self healing and protection environment*

Chapter 3 presents the evolution, design and implementation of SHAPE i.e. Self-healing and protection environment, for Computational Grid, to automatically handle the failures and the security attacks. SHAPE is a first combined self-healing and self-protection approach for complex distributed systems. It proposes an autonomic model to diagnose problems from observed symptoms, and the results of the diagnosis can then be used to trigger automated response and

recovery.

Chapter 4 *Deployment, Testing and Validation of proposed model*

Chapter 4 describes the test environment, deployment and results. Various tools used to implement SHAPE model are mentioned. Validation of the model is done using standard metrics for self-healing and self-protection. Detailed discussion on results is done.

Chapter 5 *Conclusion and Future Scope of the work*

Chapter 5 presents the conclusion of the thesis, describes the main contribution of the thesis and highlights future research direction based on the results obtained.

Chapter 2

Literature Survey & Problem Formulation

2.1 Introduction

Grid infrastructure provides the ability to dynamically link resources as one single unit to support the execution of large-scale, resource-intensive, and distributed applications. However, Grid computing comes along with a completely new level of complexity. Like in traditional distributed computing management mechanism, each resource is separately analyzed and specific parameters are adjusted for each one of them to give the best output. When trying to adapt the same procedures to grid computing, the vast complexity of the system make this task extremely complicated. To deal with this complexity, concept of autonomic computing is used.

Autonomic computing is the concept of designing complex information technology environments with the ability to perform self-management of tasks with-

out any human support. Self-management is the process which involve self-configuration, self-optimization, self-healing, and self-protection. For such systems, administrators would only be required to specify a high level requirements of the system. This greatly reduce the human requirements for administration of large systems.

Self-healing and Self-protection ultimately attempts to reduce complexity of system. Self-healing attempts to keep the system running by meeting user requirements, while Self-protection attempts to prevent the system from doing things which are not supposed to be done. One difference between self-healing and self-protection is that the success of self-healing can be improved by healing the system to fulfill requirements again. If a self-healing system can heal quickly, it can minimize visible failures to meet obligations and in self-protection, once a prohibited action can be performed, it can never be guaranteed that self-protection can recover from this action.

This chapter is organized as follows: Section 2.2 defines self-healing and discusses the need for having self-healing system for grid. Various kinds of failures are discussed in section 2.2.1. To have detail view of related work, self-healing component is further categorized into fault detection and fault healing. Details for each component are discussed in section 2.2.2. Self-protection is discussed in section 2.3.Various common attacks on grids are discussed in section 2.3.1. Existing grid computing security approaches are categorized and discussed in section 2.3.2. For implementing self-healing and self-protection, brief introduction to machine learning is described in section 2.4. Two of the best machine learning approaches are identified and mentioned in section 2.4.1 and section 2.4.2. Based upon the above discussions section 2.7 summarizes the chapter after formulating

problem statement in section 2.5.

2.2 Self-healing

Self-healing is an essential component of every computing system. It is a widely researched topic in the field of grid computing. In a grid environment there are potentially thousands of resources, services and applications interacting with each other. Since all these elements are extremely heterogeneous in nature, there are many possibilities of failures, that not only include independent failures of each element, but also those resulting from interactions between them [115][142][41][29]. The need for self-healing is especially acute for large parallel applications since the failure rate grows with the number of processors and the duration of the computation.

Self-healing is the ability of a system to recover from faults that might cause parts of system to malfunction [112]. The main objective of such systems is to reduce fault rate and keep system up and running. For a distributed system to be self-healing, it must first detect the failure and then isolate failed node from actual system. The isolated node is then repaired and reintroduced into the system. This approach of self-healing is the active approach. Self-healing systems also acts proactively. In proactive approach, precautions are taken to avoid/delay fault occurrence. This helps in reducing the failure rate. For making self-healing system a success, following must be considered [149]:

- How new nodes join the system?
- How computing resources are shared?

- How the resources are managed and distributed?

2.2.1 Kinds of failures

Need for self-healing arises when system failures start multiplying because of the large, complex and heterogeneous nature of the grid. Following are the most common categories of failures in the grid environment[120] [121] that disrupts the normal functioning of the grid.

2.2.1.1 Hardware failures

Hardware failures take place due to faulty hardware components such as CPU, memory, and storage devices[129]. Sometimes the hardware fails because components are used beyond their specifications [57]. Moreover, hardware performance degrades after two to three years. Recent studies conducted by Schroeder and Gibson [19] and Egwutuoha et al. [66], show that for of dealing with hardware failures (processors, hard disk drive, integrated circuit sockets and memory), complex distributed systems require 10,000s of disks attached to 1,000s of I/O nodes. The numbers alone imply severe problems with reliability. I/O failure and data unavailability cause significant loss to system productivity[25]. A hardware failure occurs when a hardware component is broken and needs replacement or when the system behaves abnormally and delayed job execution. It is always difficult to deal with hardware failures. Some of the common faults causing hardware failures are:

- *CPU faults*: These kind of faults arise due to faulty processors. This results into performance degradation of system.

- *Memory faults*: Those errors which occurs because of faulty memory in the RAM, ROM or cache are called memory faults.
- *Storage faults*: occur for instance in secondary storage devices with bad disk sectors.

2.2.1.2 Network failures

Network faults are the failures which are related to physical or operational faults in network [158]. Grid has complex network structure where large number of heterogeneous resources are connected over multiple domains to act as single unit. This increases the probability of network failures like packet loss or packet corruption in grid.

- *Node failure*: In node failure, individual node might go down because of any reason which results in breakage of network.
- *Packet loss*: Packet loss can occur because if broken or congested networks.
- *Corrupted packet*: Packets can be corrupted in transfer from one end to another. Corrupted packets are of no use to network administrators.

2.2.1.3 Software failures

There are several high resource intensive applications running on the grid to do particular tasks. Several software failures like the following can take place while running these applications.

- *Memory leaks*: These are an application specific problem, where the application consumes a large amount of memory and never releases it. This

causes lot of memory to be blocked even when the application don't need it.

- *Operating system faults*: These faults are mainly caused as result of applications running for grid starts interfering with operating system services. Another reason for such faults is because of deadlock or inefficient or improper resource management.

- *Resource unavailable*: Sometimes an application fails to execute because of resource unavailability. Primary reason for such faults can be resource to be used is busy with other applications. Another reason can be the resource become unresponsive because of any software exception.

- *Un-handled Exceptions*: Main reason for such failures is because of missing efficient exception handling. This includes exceptions are not properly thrown or once thrown correct action is not taken in response to exception type.

2.2.2 Classification of self-healing system

This section provides a detailed description of various approaches for self-healing system. It also includes the classification and a review of the literature as well. The Self-Healing system usually consists (Figure:2.1) of i) Fault detection and ii) Fault healing.

2.2.2.1 Fault detection

To enable self-healing distributed systems, an important component is a scalable self-detection capability. A simple technique would be that any two nodes in a

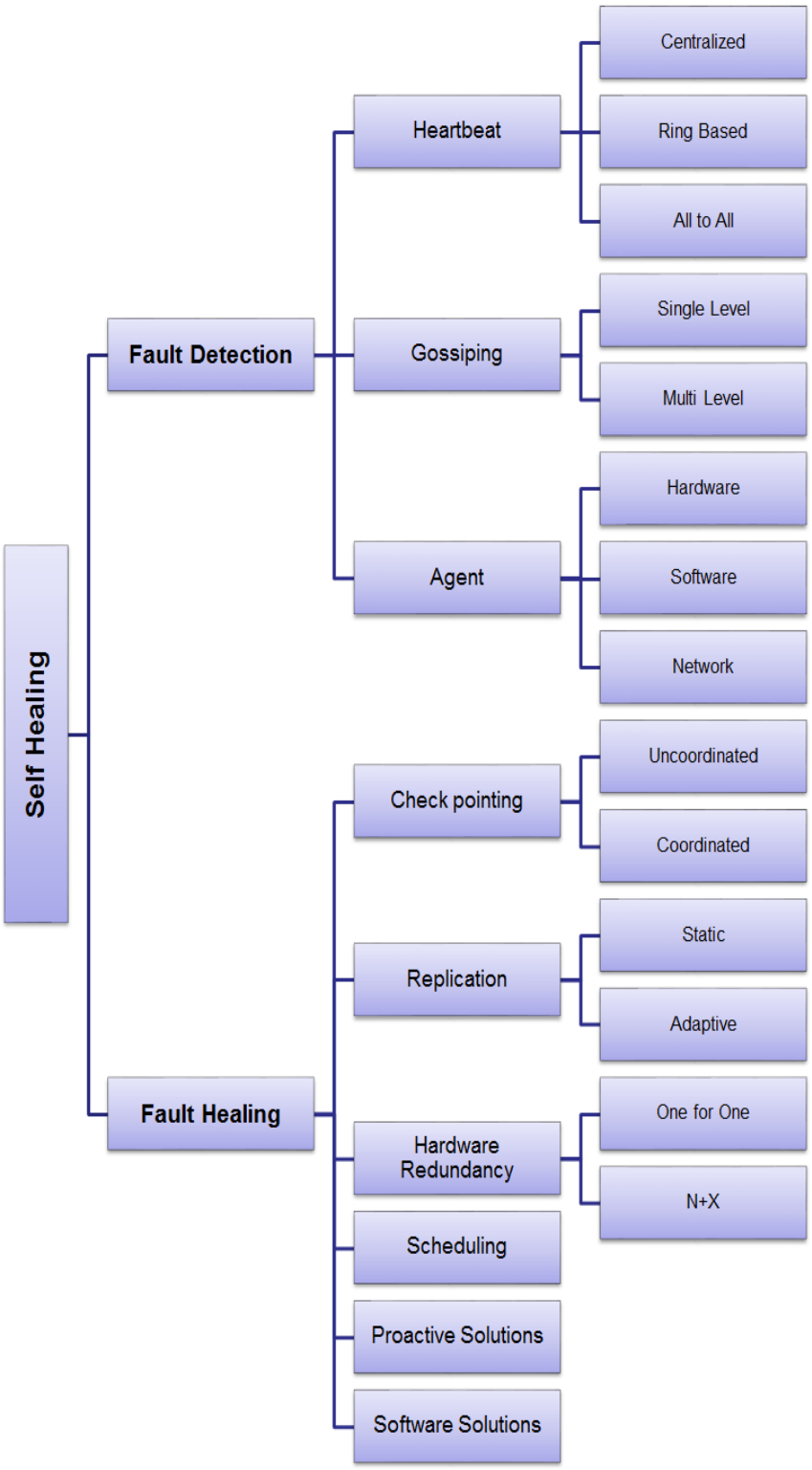


Figure 2.1: Classification of Self-healing system

distributed system monitor each other. As this results in a huge overhead and unscalable behavior, intelligent strategies are needed to manage the monitoring responsibilities. Fault detection is based upon the two models- Push and Pull Model [127] [13]. In Push model, grid components periodically send heartbeat messages to failure detector, announcing that they are alive. In the absence of any such message, the detector considers some failure has occurred at grid component level. In the pull model, it is failure detector that asks for the node status from grid components(Figure:2.2).

Fault detection has been further categorized into following categories:

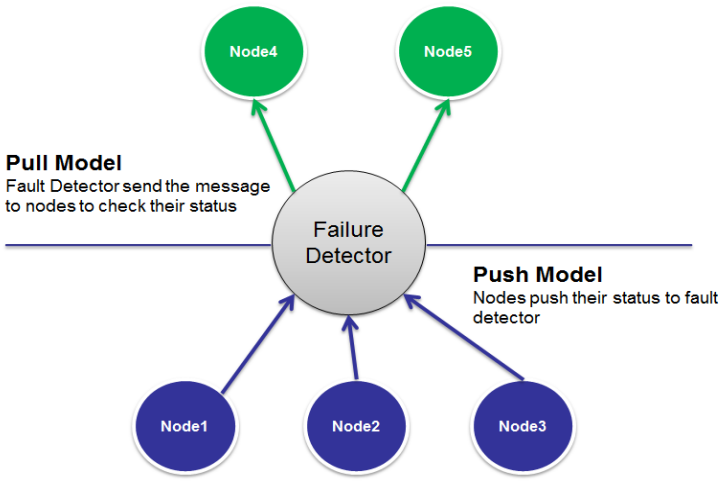


Figure 2.2: Push-Pull model

- Heartbeat Monitoring: Heartbeat technique is the base technique which most of the detection units follow. The heartbeat techniques [9] are further classified into 3 types (Figure:2.3):
  - *Centralized heartbeating*: In centralized heartbeating, heartbeat signals are sent to one central node which creates a hot spot to monitor

all nodes. That central node is responsible for providing status for all the nodes associated with it.

– *Ring based heartbeating*: In this type of heartbeating, each node monitors the next node and ring based structure is formed. Drawback of this type of heartbeating is that a virtual ring suffers from unpredictable failure detection times when there are multiple failures, an instance of the perturbation effect.

– *All-to-all heartbeating*: This involves sending heartbeats to all members. Major drawback of this technique is that it causes the message load in the network to grow quadratically with group size, again an instance of high asymptotic complexity

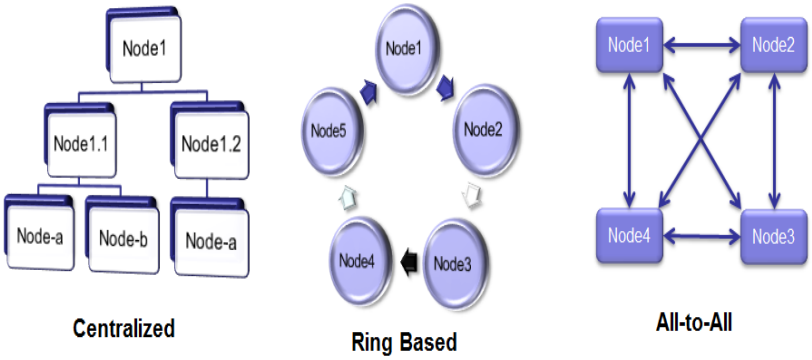


Figure 2.3: Heartbeat techniques

Another approach called "Application Heartbeat" [47] is also trending. Its goal is to manage the performance of software applications which have been instrumented to emit their performance level via the application heartbeat framework [103]. By making calls to the heartbeat API, applications signal "heartbeats" at some important places in the code. Additional functions in

the heartbeat interface allow applications to specify their goals in terms of a desired heart rate [102].

Globus GT4.0, one of most widely used grid middleware, also uses a heartbeat technique to monitor running processes and to detect faults [114]. Once failure is detected, the application is notified of the failure and appropriate recovery action is taken manually. Globus Heartbeat Monitor (HBM),comprises of three components:

- *Local monitor*: This is responsible for monitoring the computer on which it runs, as well as selected processes on that computer.
- *Client registration API*: This API helps the application to specify the processes to be monitored by the local monitor, and to whom heartbeats are sent.
- *Data collector API*: Enables an application to be noticed about relevant events concerning monitored processes.

- Gossiping: It deals with the process of transferring information within the distributed systems by randomly choosing nodes for communication. The first gossiping approach was given by Renesse et. al. [130] whose main goal is to attain the scalability of the system. This approach is based upon the basic approach where each node maintains the list of heartbeat counter for known processes. Randomly each process increments its counter and sends the list to other processes. The increase in counter shows that the process on a node is alive. This list once received by the other node, then the receiving node updates its list based upon the current counter values in the list. If the list is not updated for certain threshold time, the process is considered

as dead. The main drawback of this approach is the random selection of node. If certain node is not picked for certain time, it is regarded as failed.

The approach is extended for distributed systems by using multi-level gossiping algorithm. This approach optimizes way of randomly choosing nodes. For this they concentrate on the traffic within subnets. This increases the scalability of system but the major drawback is that with the increase in number of processes within the subnets, gossiping message also increase. This reduces the performance of the system.

- Agent based: An agent-based approach helps to implement proactive approach for dealing with faults. Different agents can be set up to monitor system and gather information like memory consumption, resource availability, hardware information, and various network related information. This information is then used to improve efficiency and reliability of grid services [146]. A good example of such architecture is defined in Grid Architecture for Computational Economy (GRACE)[[126], [125]]. GRACE manages extra-functional software properties such as reliability and fault tolerance in complex component based software architectures.

2.2.2.2 Fault healing

Fault healing deals with the process to recover the system from failures. This can be active or proactive process. In active fault healing, action is taken once fault has occured whereas in proactive process, precautions are taken to reduce the failures. Various techniques for fault handling are discussed below.

- Checkpointing: It provides an effective technique for tolerating resource

failures which occurs for short-duration [37]. It aims to avoid a total loss of results by saving the state of executing program at different time intervals and re-executing the program from the last working state whenever failure is encountered [67]. Checkpointing distributed applications is more complicated than checkpointing the non-distributed applications. While capturing the checkpoint for the distributed application, the system should capture the details for all the individual processes running for that application. Along with this, all the communication channels associated with the processes should also be captured. Few of the examples where checkpointing is used in grid middleware are:

HTCondor (previously called Condor) provides system-level checkpoints for applications running on condor environment. Checkpointing in condor doesn't give best results for high-performance parallel programs [133][49]. Condor-G [69] is having better checkpointing support as compared to condor. It has capability to safeguard the system from four kind of failures. This includes- Job manager crash, crash of components like GateKeeper and JobManager which manages the resources, GridManager crash and network failures.

N1GE6 uses 2 level of checkpointing [116]:

- *Kernel-level*: This type of checkpointing uses the operating system kernel to create checkpoints for complete process space and store it on physical location.
- *User-level*: For user level checkpointing, a program is embedded into the application which periodically updates the status of application to

physical storage.

Checkpointing [133] is basically divided into 2 types:

- *Uncoordinated Checkpoint*: This checkpointing technique is distributed in nature. Here each process running in the system maintains the information about the checkpoints. Whenever the recovery for certain application is done, checkpoints from each process are clubbed to form a global checkpoint. Uncoordinated checkpointing is not suitable for systems with large number of running processes. This is because of the unbounded rollback propagation during recovery.
- *Coordinated Checkpoint*: This type of checkpointing uses the global checkpoint for recovery. The global checkpoint is formed by arranging various individual checkpoints. The advantage of coordinate checkpoint over uncoordinated checkpoint is that it reduces the storage overhead.

  Many approaches and algorithms are proposed and implemented for checkpointing and rollback. For achieving coordinate checkpointing, "A Faster Checkpointing and Recovery Algorithm" is proposed by Wen Gao et.al [45]. This provides rollback recovery mechanism and watch-dog timer detector for fault tolerance. Faster rollback recovery algorithm named diskless checkpointing and rollback is proposed to reduce the disk write overhead and thus improving the performance. Drawbacks of this algorithm are: the cluster cannot tolerate multiple transient failures using this diskless checkpointing algorithm, and is

more memory intensive. In the worst condition, the requirement will be 3 times of the checkpointing data size. At the same time, application running time increases due to the memory writing of the faulty page.

Jiang et.al [119] brings the concept of Communication Induced Checkpointing (CIC). CIC protocols allow processes to perform dual tasks-to take checkpoints and simultaneously check that each checkpoint taken is useful and satisfy the checkpoint-inducing condition. The major drawback of CIC is the high overhead for creating checkpoints. When the number of processes increase in complex systems like grid, this causes performance of the system to degrade.

Large overhead problem with CIC is resolved by protocol based approach called HOPE [154].Hybrid optimistic checkpointing and selective Pessimistic message logging (HOPE) protocol, which tries to achieve the balance between checkpointing, message logging overhead and scalability using group based strategy. Inside each group, HOPE uses a CIC protocol, called Optimistic Checkpointing and Message Logging approach [119], to create consistent global checkpoints. XtreemOS grid checkpointer is another approach designed and implemented by Feller et al. [34]. This is based upon the independent checkpointing based approach.

Few of other known approaches used in grids for checkpointing are-Team based message logging by Meneses et al. [35] and correlated set based protocol approach by Bouteiller et al. [5].

- Replication: It always works on assumption that the probability of a single resource failure is much higher as compare to simultaneous failure of multiple resources. The replication technique runs different replicas of same task on different grid resources simultaneously, hoping that at least one of them will complete successfully[144]. Unlike checkpointing, the replication avoids task re-computation by executing several copies of same task on more than one compute stations.

  Replication techniques are categorized into static and adaptive. In static replication number of replicas for the task are decided prior to execution and are fixed. This approach most of the time increases load on grid and became the reason for over utilization of resources.

  Adaptive replication on another hand decides optimal number of replicas needed for correct execution of tasks. For implementing adaptive replication many approaches exist. K. Srinivasa, G. Siddesh and S. Cherian [86] proposed an middleware for adaptive replication based upon data replication at different sites of the grid. The middleware dispatches replicas to different nodes and enables data synchronization between multiple heterogeneous nodes in the grid using TCP/IP transfer protocol.

  M. Amoon [95] considers adaptive job replication technique in order to create a proactive fault-tolerant scheduling system. Two algorithms are proposed for this. First algorithm is for determining the number of replicas for each job and second algorithm is used to selecting the resources that execute these replicas. Both these algorithms consider fault rate to dynamically determine number of replicas.

The main disadvantage of replication technique in general is the additional resources used in executing the same job. This can cause grid over provisioning and can lead to great delays for other jobs waiting these resources to become free[144].

- Hardware Redundancy: With hardware redundancy, redundant nodes are added to make it possible for the systems to tolerate failures [128][48][27]. When one of the component fails due to any reason, there is always another spare component that can take its place. In case of failures, remove the faulty component and bring the backup component on its place to work. Actual component once repaired is again added to the system. To have the hardware redundancy its true meaning for fault tolerance, software dependencies are also there. By using reactive software techniques like checkpointing and restart, efficiency of the system is further improved. In general their are two main approaches for implementing hardware redundancy- "one-for-one redundancy' and "N + X redundancy'. In "one for one approach', equal number of backup nodes are kept as that of actual working nodes in system. The main drawback here is that it doubles the hardware cost. N + X approach is used to overcome the drawback of one-for-one approach. In this approach, the backup nodes are less as compared to the actual node.

- Scheduling based: Scheduling acts as the another way to deal with failures. In the complex systems like grid, deploying the support for integrated fault-tolerance with scheduling approaches is of paramount importance [78]. The basic technique used in most of the fault tolerant scheduling algorithms is the coupling of scheduling policies with the job replication schemes such

that jobs are efficiently and reliably executed. Figure 2.4 shows the generic scheduling based fault tolerance technique.

Many approaches are proposed by researchers for using fault-tolerance with scheduling. Distributed fault-tolerant scheduling (DFTS) algorithm [78] is one of well known algorithm that uses the coupling of job scheduling with job replication. This approach is based upon the assumption that grid is divided into different sites. Each site is controlled by respective site manager. Managers from different sites can communicate and can act as the backup resource for other manager. The algorithm uses fixed number of replicas for each job. Each job replica is scheduled to a different site to be executed. The number of replicas is specified by the user at the time of job submission.

Another approach for using replication with scheduling is given in [18]. Here the authors Khoo and Veeravalli proposed a replication-based fault tolerant algorithm which schedules jobs by matching the user security demand and resource trust level. The number of job replications changes adaptively with the security level of the grid environment.

Chtepen et.al [100] presented some scheduling heuristics based on task replication and rescheduling of failed jobs. Their heuristics do not depend on particular grid architecture and they are suitable for scheduling any application with independent jobs. Scheduling decisions are based on dynamic information on the grid status and not on the information about the scheduled jobs.

- Proactive solutions: These solutions uses past history of resources with ref-

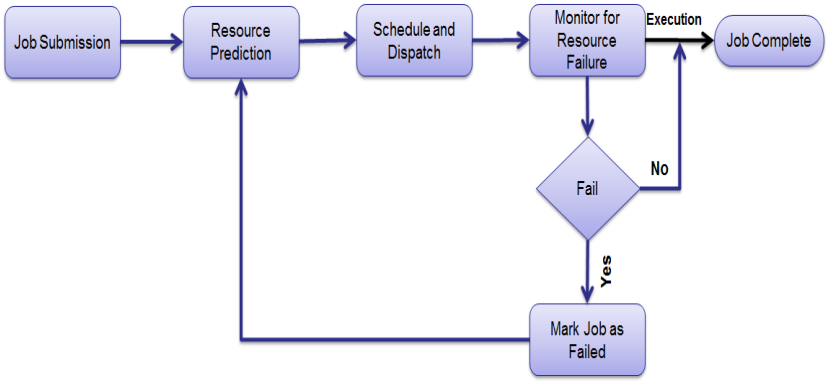


Figure 2.4: Basic schedule based fault handling

erence to the service level aggrement (SLA) for reducing the failure rate. The AssessGrid project [82] proposes a model to estimate the probability of SLA failures in Grid environments, and considers the probability of n resources failing for the scheduled duration of a task as well as the probability that m reserved resources are available for that duration. The final probability for node failure is based upon the assumption that node failure represents a Poisson process which is non-homogenous in time. This project is further used to have [83] SLA brokering mechanism with risk assessment support. WS-Agreement and risk metrics are used to facilitate SLA creation between service consumers and providers within a typical Grid resource usage scenario. During job execution the provider can use risk management techniques to fulfil SLAs. Fewer violated SLAs will lead to improved performance. Thus brokers will be able to estimate the reliability and trustiness of resource providers. The limitations of this project seems to be the overestimate/underestimate of failures risks.

Recently, a mathematical model to predict the risk of failure of resources in such environments using a discrete-time analytical model driven by re-

liability functions fitted to observed data [121]. The model relies on the resource historical data so as to predict the risk of failure for a given time interval. This model thus helps in ranking the grid resources based upon the respective risk factor. Major drawback with this model is that it takes very basic attributes to do the analysis of resource based upon its history. Due to this the results are not always right and also if the resource is new prediction can't be done correctly.

- Software Solutions: These solutions mainly rely upon the software to monitor failures. Two of the main solutions based upon software techniques are given by Cheng et. al [143] and Huns et. al.[108]. Cheng and team has proposed a self-adaptive system using software architectural models. Software architectural models are maintained at runtime and used as a basis for system reconfiguration and adaptation. Their system uses three layer architecture comprises of Runtime Layer, Model Layer and Task Layer.

  The approach given by the Huns and team on the other hand is based upon the multiagent systems that are used to achieve robust software.The model is based upon the fact that algorithms, in the form of agents, are easier to reuse than when coded conventionally and easier to add to an existing system, because agents are designed to interact with an arbitrary number of other agents.

2.3 Self-protection

A self-protecting system proactively detects and identifies arbitrary attacks and takes independent actions to protect itself against intrusive behavior. Its main aim is to protect grid environment against deliberately planned malicious activities. The self-protecting systems can also passively scan suspicious events without users being aware of that such protection is in place. [17].

2.3.1 Security threats in grid

Although grid security system provides fundamental security mechanisms like authentication, authorization, confidentiality. Grid web services also provides security systems like WS-security, WS-conversations, but still there are many other possible security threats associated with grid environment. Various kind of security attacks against which self-protection is needed are:

2.3.1.1 QoS violation

The grid is envisioned to provide the best quality service respecting the service level agreement (SLA) signed between the user and the service provider [131]. Then also the quality of service is being violated by attacks. Following are some common QoS violation attacks:

- *Dropping Packets:* In these types of attacks, the attacker drops the packets flowing through the grid networks by compromising some grid component in that network. For eg: compromising the router and reprogramming it to

discard some specific network packets. This packet dropping attack targets the availability of the system.

- *Delaying Packets:* In this attack, the attacker targets flow rate of the packets. The attacker decreases the overall network traffic flow rate to reduce the effective QoS.

2.3.1.2 DoS/DDoS attacks

DoS attacks are massive attacks. They are launched by a set of attackers that generates huge traffic to flood the victim's network, with the aim of making the computer or network resources unavailable to its intended users. For e.g. The Sun Grid was forced to come down by a distributed denial-of-service (DDoS) attack [73], which later required an emergency login procedure change. While grid computing may very well transform enterprise computing, but such incidents underscore the security risks that could prove quite distressing for enterprises that rely on grid computing. Some of DoS attacks include:

- *TCP floods:* In this type of attack, a stream of Transmission Control Protocol (TCP) packets is targeted towards the victim machine with various TCP flags set. The packets are spoofed to appear as legitimate connection request. The spoofed packets are received at the server end but TCP three-way handshaking never completes. The server, on the other hand, tries to answer the incomplete TCP connection. After several malformed packets are sent to the server, the server may stop responding to legitimate connections until it has resources available to process the additional requests. In TCP floods mostly SYN, ACK, and RST flags are used.

- *ICMP echo request/reply (Smurf Attacks)*: Smurf Attack is another DoS attack. In this the attacker broadcast a large number of Internet Control Message Protocol (ICMP) "echo-request" packets with victim's spoofed source IP address in the network. As the destination IP contains broadcast address of the network, all the machines in the network will receive the "echo-request" packet and then they all will try to respond with a "echo-reply" message to the victim machine. Now if the number of machines in a network is quite large then victim's machine will be flooded by reply messages.

- *UDP Storm*: This attack exploits two User Datagram Protocol (UDP) connection services. One is a character generation ("chargen") service which generates a series of characters each time a UDP packet is received. And the other is "echo "service which echoes a character that it receives. The attacker takes advantage of this and sends a spoofed packet with victim source IP address to another machine. The exploited services start responding and the two machines get engaged in useless communication thus burdening the network.

- *IP spoofing*: IP spoofing is a way to impersonate or conceal the identification of the sender or another computing machine. In this a false source IP address is placed in the IP packet. IP spoofing is used in variety of network attacks and most frequently in DoS attacks. In DoS the attacker need not to worry about attack responses as they all will be directed towards spoofed IP address. Also IP spoofing is difficult to detect, thus hiding the true source.

2.3.1.3 Insider attacks

Insider attacks are performed by potential users with authorized system access. These attacks are more dangerous and difficult to detect than external attacks [65] because the intruders inside the organizations have more privileges and are aware of the network architecture, network vulnerabilities and network security policies and procedures. Being a trusted member of the organization, the culprit has authorization to observe and work with organization's IT infrastructure [54]. These attacks typically target specific information and exploit established entry points.

- **Remote to local(R2L)** Attacks in which an unauthorized user can bypass normal authentication and execute commands on the target [101].
- **User to root (U2R)** Attacks in which a user with login access is able to bypass normal authentication to gain the privileges of another user, usually root. This can also be the case when the user tries to use more or some other resources not assigned to him [101].

2.3.2 Classification of grid security systems

2.3.2.1 System level

- Information Security: This type of security consists of:

  Authentication: It is usually linked close together with authorization. Authentication and authorization are often used in a combination in order to grant someone access to a service or a resource based upon a given identity. In both [57] and [58], authentication is pointed out as a distinct

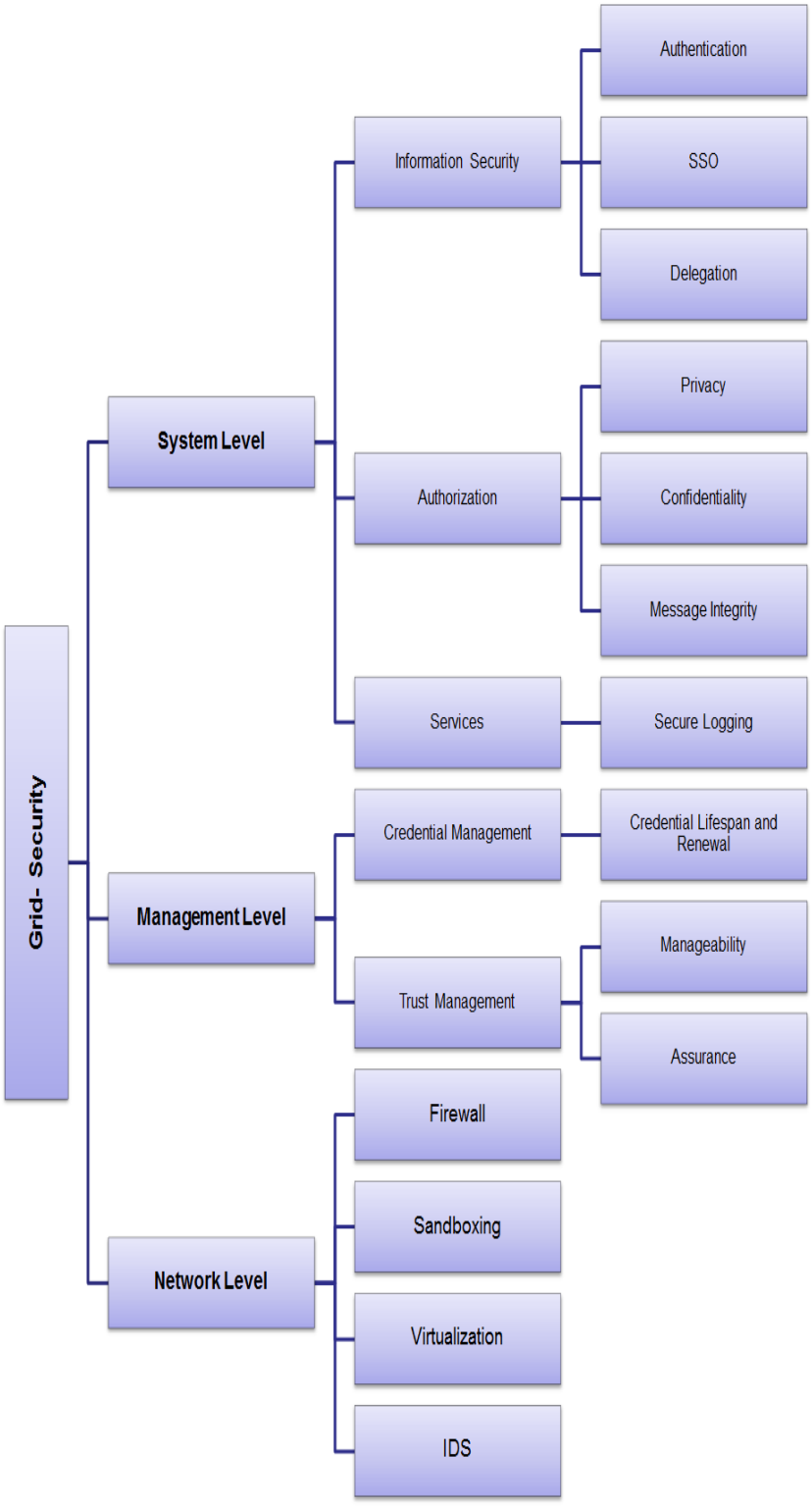


Figure 2.5: Grid security classification

mechanism with the purpose of verifying proof of an asserted identity. The authentication mechanism in a Grid Computing Environment(GCE) is to provide plug points for the multiple authentication mechanism at hand, and the means for conveying the specific mechanism.

In [40], it is stated that in order to get a strong authentication mechanism single sign-on is needed. This is because multiple authentication requests are bothersome and will likely be circumvented if possible. Web services security (WSS) [16] is a security structure designed for systems similar to a GCE. In WSS mechanisms that may be used for authentication in a GCE are already implemented, such as Public Key Infrastructure [11], [150].

Single sign-on: It is needed because users participating in a GCE often need to coordinate multiple resources just to solve one single task. Manually performing an authentication process in such a scenario would be overly burdensome. A security mechanism is needed to ensure that the entity having successfully completed the act of authentication once, wont need to re-authenticate in a given period. One must remember that requests may span several security domains and should hence be a factor between authentication domains and mapping of identities. Because of this, delegation of an entities rights and the ability to indicate the identity of intermediate entities is needed.

Delegation: The virtual organizations (VOs) in a grid underlying collaborative work, may form quickly, evolve over time and span organizations [57]. The effective operation of these VOs depends critically on trust. One solution to this is establishment of dynamic trust domains where one entity can assign rights to another. To manage this, a delegation service is

needed such that authority can be delegated from one entity to another. Delegation is also needed to secure dynamic service creation [57], [23]. This mechanism/discipline is also recognized in [62].

- Authorization is usually closely linked to authentication (e.g. authentication is needed to access services, which the entity is authorized to use). To access specific services in the grid, one needs to be authorized to access that service first. In a grid, authorization policies work both ways (not only as in the basic model where policies are being specified by the resource owner). This is because requestors may need the provider to fulfill some requirements. Policies for authorization should also mention if mutual authentication is needed [92]. Authorizations helps in providing:

  Privacy: Both service requestor and provider must be allowed to define and enforce privacy policies, taking into account personally identifiable information for the purpose of invocation. In [57] it is stated that privacy policies may be treated as an aspect of authorization policy addressing privacy semantics such as information usage rather than plain information access.

  Confidentiality: Both the underlying communication mechanism and the messages or documents flowing over this given transport mechanism should preferably be confidentiality protected. If only the transport mechanism were protected, the information might be unprotected for a short time while on transportation endpoints in the grid. If the message has to go through a computational facility, the transport layer will probably decrypt it, and then encrypt another time before the message is forwarded. Because of

this message encryption is also needed. This means that confidentiality requirements include point to point transport as well as store and forward mechanisms. The need for communication security such as confidentiality is also pointed out by [23], [92], [56].

Message integrity: Both confidentiality and non-confidentiality protected information can be altered. To protect against unauthorized changing of information in messages/documents some kind of integrity protection is needed. Preferably the transport mechanism should at least have integrity protection that guards against transmission errors, but also against intended but unauthorized altering of the information. Using integrity and confidentiality protection can help in achieving communication security [23], [92], [56]. Using integrity protection at the message/document level is often subject to policy and quality of service requirements.

- Services

  Secure logging: Logging is important to make a foundation for addressing requirements for notarization, non-repudiation, and auditing. This logging should be performed in a secure manner, or else this logging cant be trusted. Logging should include secure logging of any kind of operational information or event since this can be used for auditing. Logging in a secure manner means reliably and accurately, which means so that such logging is neither interruptible nor alterable by an adversary.

2.3.2.2 Management level

- Credential Management

  Credential life span and renewal: Credentials have to be renewed after a given period. This is to limit the risk of compromise in delegation and single sign-on [57], [151], [56]. Different tasks in the GCE will have different lifespan and execution time. Execution time for performing the same task can vary because of resource usage from other services in the grid. Because of this, it will not always be possible to predict the precise credential lifetime needed for the task. A user needs to be notified or have the possibility to refresh his credentials if a task takes longer time than the lifetime of his initial credentials.

- Trust Management

  Manageability: The ability to manage security in a grid is needed. The fact that a grid needs authentication and authorization indicates that both identity and policy management are needed. This management also includes higher-level requirements such as virus protection, intrusion detection and prevention. Virus protection and intrusion detection are requirements on their own, but are typically provided as part of security management.

  Assurance: Means to qualify for the security level expected of a hosting environment, must be provided by every participating node. This includes what security measures and mechanisms are implemented, and policy of their usage. This can be virus protection, firewall usage for internet access, and internal virtual private network usage [87].

2.3.2.3 Network level

Network level techniques deals with research focused on network based solutions for securing grid. Proposed solutions falling in this category focus on protecting the grid resources including grid nodes and communication network.

Firewalls are major barriers to dynamic and cross domain computing in general, and also to cross domain grid computing [42]. Firewalls might only be of minimal value in an environment that carries out dynamic cross-domain computing, but firewalls are unlikely to disappear anytime soon [57]. Because of this, a grid must take firewalls into account so that they can be traversed securely without compromising local control of firewall policy.

Sandboxing and **Virtualization** comes under this category of isolation techniques to protect grid nodes [4] from attacks. The best known sandboxing technique is called Entropia [7]. Entropia is kind of virtual machine designed specifically for desktop grids. It helps to protect applications, clients, processes and resources on the grid. Virtual Private Grid infrastructure (VPG) [38] is another way of using virtualization for isolating nodes from attacks. This involves harnessing virtual private network technology and applying it to grid computing. This infrastructure works around heterogeneous, locally-specific security.

Intrusion detection Systems (IDSs) are mostly used to provide system level security. These can be configured to detect intrusions within complex systems like grid. IDS raise an early alert for any suspicious activity that occurs during and before the attacks. These alerts are then processed by network administrator. IDS systems specifically related to grid computing are discussed

below.

To provide additional level of security to grid, many IDS's are designed and developed [28]. IDS first function is to monitor the network for any abnormal activities. Another component of IDS is sensors which capture all information and pass it to analyzer. For manual analysis, the administrator analyzes alert reports and takes some actions[98]. Some analyzers also use system like snort [141] for analysis purpose.

Schulter et al. [14] describes different types of IDS systems and identify the need for IDS for grids. Based upon their analysis, they have proposed a high level GIDS that clubs the functionalities of host based and network based intrusion detection systems. SANTA-G (Grid enabled System Area Networks Trace Analysis) [132] is a system whose core strength is in querying the log files through Relational Grid Monitoring Architecture (R-GMA). SANTA-G queries snort alert logs from SQL to start its processing. SANTA-G is composed of three elements: A Sensor, a QueryEngine and a Viewer GUI.

Another example of grid based IDS is GIDA [107] which also uses a similar structure. IDS on Oracle lOG database is provided in [3]. IACID [145] from USC, provides a Grid based IDS system having separate network and host IDS systems. Michal Witold [106] in his thesis investigated the possibility of Grid-focused IDS. The main stress has put on feature selection and performance of the system. Leu and Li [39] proposed Fault-tolerant Grid Intrusion Detection System (FGIDS) which exploits grid's dynamic and abundant computing resources to detect malicious behaviors from a massive amount of network packets. In FGIDS, a detector can dynamically leave or join FGIDS anytime.

Snort [118] is the most commonly used signature-based detector that runs

over IP networks analyzing real-time traffic for detection of misuse. Snort also provides the option to make it work as anomaly detection IDS by using the preprocessor component. Based upon these two approaches, many approaches are proposed to handle intrusions.

M. Ali Aydin et.al [93], proposed hybrid IDS by combining the two approaches in one system. The hybrid IDS is obtained by combining packet header anomaly detection (PHAD) and network traffic anomaly detection (NETAD) which are anomaly-based IDSs with the misuse-based IDS Snort. The results shows that hybrid IDS is more powerful than the signature-based. The problem with this approach is that its performance will degrade when the traffic on the unit running IDS increases. This is because it installs the IDS on single unit that will work for single network but for distributed network, this approach is not good.

Yu-Xin Ding et al [147] proposed another Snort-Based Hybrid IDS. It is divided into three modules: misuse detection, anomaly detection and signature generation module. Snort is used as misuse detection module to detect known attacks. Anomaly detection module uses Frequent Episode Rule mining algorithm with a sliding window to generate rules for Anomaly detection. Signatures of newly detected attacks by Anomaly detection module are generated by using Signature generation module. It uses Apriori algorithm for signature generation. It provides good performance in offline detection, but cannot be used for real time detection.

J. Gomez et al [70] made another attempt to use the snort preprocessor capability to design system called Hybrid IDS i.e. H-IDS. In this, basic statistical method uses moving averages corresponding to network traffic. This is a very basic model that tends to use some data mining techniques to predict the future

performance of the system.

Vijay Katkar and S. G. Bhirud [148] proposed a light-Weight mechanism to detect novel DoS/DDoS (Resource Consumption) attacks and automatic Signature generation process to represent them in real time. Condition based network connection records omission used for Novel attack Signature Generation increases the speed and accuracy. Limitation of this technique is that this is only limited to attacks that are related to resource consumption. No other attack like DoS, DDoS, R2L and L2R attacks are taken care of. Other similar researches done on snort includes [32][84][91]. For instance, [32] models only the http traffic, [84] models the network traffic as a set of events and look for abnormalities in these events, [91] enhance the functionalities of Snort to automatically generate patterns of misuse from attack data, and the ability of detecting sequential intrusion behaviours, [94] that is a pre-processor based on studying the de-fragmentation of package in the network to avoid evasive attacks in the IDS.

All the techniques discussed use centralized system. The major drawbacks of all such systems are high rates of false positives, low efficiency, etc, especially in case of distributed attacks. Many distributed agent based techniques are also developed to handle all these drawbacks. Imen Brahmi et al[50] proposes technique called DIDMAS (Distributed Intrusion Detection using Mobile Agents and Snort) that focuses only on misuse detection approach. The experimental results show the effectiveness of this approach and highlighted the DIDMAS realizes the scalability of mobile agent based approaches as it reduces bandwidth consumption and also response time. The main drawback of this approach is that it is not capable of detecting any new attacks. It can detect only those attacks which are present in its signature database. One more similar approach is DIDS

(Distributed Intrusion Detection System) [46] that works on the same path as DIDMAS.

There is one more concept given by Yang et. al. [156] that uses all these network level components as a single unit to protect grid environment from different network attacks. Honeypots [109][80] are used to protect the grid from malicious activities. This model states that grid resource managers will detect the malicious activities and then redirect that traffic to honeypots/honeynets[117] (collection of honeypots form honeynets) for further analysis. No details are presented that how practically resource managers will decide the activity as malicious. Not much work is further done in this area as honeypots require lots of manual analysis on the captured data to understand the behaviour of attack. In large systems like Grid, it's not feasible to monitor all the data.

2.4 Machine learning

From the above discussion on existing fault tolerance and security techniques, need for self-healing and self-protection techniques is felt. These techniques should perform their operations in reasonable time and should be accurate enough to take independent decision without any human intervention. To achieve this goal, machine learning techniques have been found to be very useful. Machine learning systems have a basic characteristic of learning from examples and adapting themselves according to environment [137][22]. This especially suits the grid environment since the system is highly dynamic and multi-organizational.

Machine learning [157] [33] based approaches achieve effectiveness with less manual intervention and are more adaptive to different failures and continued

changes in security patterns. Furthermore, they do not depend on any predefined fault handling techniques and security detection rule sets analogous with non-machine learning counterparts. In particular, machine learning is defined as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty[88].

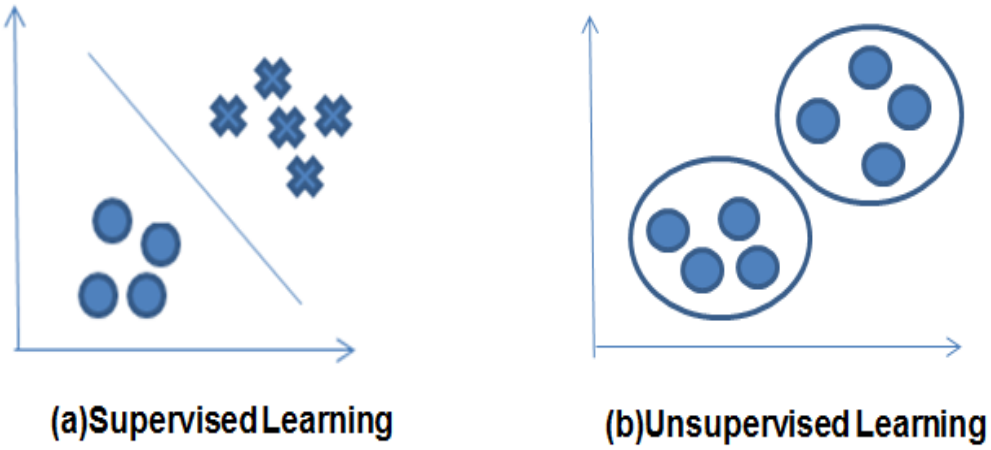


Figure 2.6: Supervised vs. Unsupervised learning

Two principal machine learning approaches for dealing with failures and security attacks in a semi or fully autonomous fashion are commonly considered, namely supervised[33] and unsupervised[159]. The former depends on an initial training set to assert classification, while the latter does not, employing rather other techniques, such as clustering, to achieve its objectives. In supervised learning, model input observations, more commonly referred to as labels, are associated with corresponding outputs upfront; in non-supervised approaches, any observations are associated with latent or inferred variables. Figure 2.6 displays

the supervised and unsupervised learning. In supervised learning input is divided based upon the label, where as in unsupervised learning there are no labels, but still the input is divided into two groups. The main idea in unsupervised is the fact that there are natural properties that help us discriminate different type of data sets. Few examples of supervised algorithm[1] are K-Nearest neighbors, C-4.5 and Support vector machine and for unsupervised learning[2] few of the approaches like clustering, blind signal separation and self-organizing map exists. Table 2.7 shows the comparison between different supervised algorithm.

	Decision Trees	Neural Networks	Naïve Bayes	kNN	SVM	Rule-learners
Accuracy in general	**	***	*	**	****	**
Speed of learning with respect to number of attributes and the number of instances	***	*	****	****	*	**
Speed of classification	****	****	****	*	****	****
Tolerance to missing values	***	*	****	*	**	**
Tolerance to irrelevant attributes	***	*	**	**	****	**
Tolerance to redundant attributes	**	**	*	**	***	**
Tolerance to highly interdependent attributes (e.g. parity problems)	**	***	*	*	***	**
Dealing with discrete/binary/continuous attributes	****	***(not discrete)	***(not continuous)	***(not directly discrete)	**(not discrete)	***(not directly continuous)
Tolerance to noise	**	**	***	*	**	*
Dealing with danger of overfitting	**	*	***	***	**	**
Attempts for incremental learning	**	***	****	****	**	*
Explanation ability/transparency of knowledge/classifications	****	*	****	**	*	****
Model parameter handling	***	*	****	***	*	***

**** stars represent the best and * star the worst performance

Figure 2.7: Comparing learning algorithms[139]

Apart from these two approaches another approach used in machine learning is Reinforcement learning. This is useful for learning how to act or behave when given occasional reward or punishment signals [88]. In other words Reinforcement learning is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment [90].

Based upon the literature survey done, SHAPE uses supervised learning and reinforcement learning algorithms. Support vector machine (SVM) is used in SHAPE from various supervised learning algorithms available [139].

2.4.1 Support Vector Machine (SVM)

SVM are among the best supervised machine learning methodology which consists of associated learning algorithms that interpret data and recognize patterns among them [122]. SVM are used for classification and regression analysis. SVM based approaches have also been proved to perform consistently well for detecting intrusions [135][138][155][36]. In grid computing, [72] uses SVM to detect intrusions. This is the only available approach in literature where SVM is used for securing grid from attacks. The proposed model uses SVM to automatically detect new DDoS attacks and uses Concentration Tendency of Network Traffic (CTNT) to analyze the characteristics of network traffic for DDoS attacks.

The basic SVM takes as input a set of data and for each given input predicts, which of two possible classes will be the output, making it a non-probalistic binary linear classifier[12]. SVM is a mathematical entity, maximizing a particular mathematical function with respect to a given collection of data. The essence of SVM classification needs only to grasp four basic concepts [153].

- The separating hyper plane
- The maximum margin hyper plane
- The soft margin
- The kernel function

2.4.1.1 The separating hyper plane

Consider the Figure 2.8, in which X's patterns represent positive training examples and O's patterns represent negative training examples, the separating hyper plane(called decision boundary) is also shown and three points have also been labeled A,B and C [12].

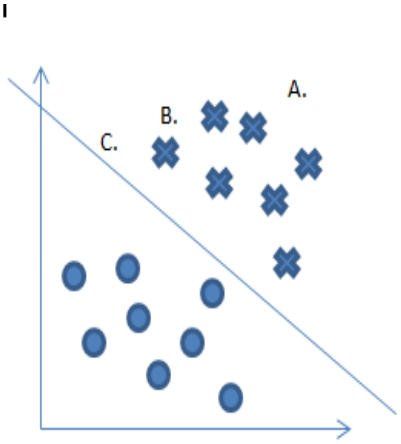


Figure 2.8: The separating hyper plane

Point A is very far from decision boundary and prediction would be made for the value of y at A as y=1. Conversely, the point C is close to decision boundary, and because it's on the side of decision boundary on which we would predict y=1, it seems likely that just a small change to the decision boundary will cause our predicted value to be y=0. We have more confidence about our prediction at A than at C. The point B lies in the middle of these two cases, and broadly, it is observed that if a point is far from the separating hyper plane, then we may significantly be more confident in our predictions. This drags us to the notion of maximum margin hyper plane.

2.4.1.2 The maximum margin hyper plane

The goal of the SVM is to identify that line which separates the different types of inputs. However many such lines exist and goal is to choose which one among them is the best classifier. The theorem from the field of statistical learning theory supports the choice that if the distance from that separating hyper plane is defined to be the nearest expression vector as the margin, then the SVM tries to select the maximum margin hyper plane [74].

2.4.1.3 The soft margin

Not always the data is linearly separable(Figure 2.9). Many real data sets cannot be separated just using a straight line as the data set contains an error. SVM must deal with errors in data by allowing a few anomalous data values to fall on the "wrong side" of the corresponding separating hyper plane. To handle such cases, the SVM algorithm is modified by adding a "soft margin". Essentially, this allows some of the data points to push their way straight through the margin of the separating hyper plane without causing any effect to the final result. SVM should not be allowed to cause too many misclassifications. Hence, introducing the soft margin also necessitates introducing a user specified parameter that controls, how many data profiles are allowed to violate the separating hyper plane and how far from the line they are allowed to go. Setting this parameter is difficult by the fact that we still want to try to have a large margin with respect to the examples which are correctly classified. Hence, this soft margin parameter specifies a trade-off between hyper-plane violations and size of the margin.

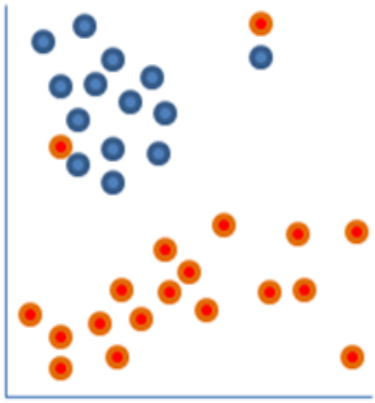


Figure 2.9: The soft margin

2.4.1.4 The kernel function

The kernel function adds an additional dimension to the dataset. It is just a mathematical trick that allows the SVM to perform a "two-dimensional" classification of a set of data which was originally one-dimensional. A kernel function[77] projects the data from a low-dimensional space to space of higher dimension. If we choose a good kernel function, then the data will become separable in the resulting higher dimensional space (Figure 2.10). It is easy to prove that for any given data set with consistent labels, there is a kernel function that will allow the data to be linearly separated. However projecting into very high-dimensional spaces can cause problems, due to the famous curse of dimensionality because as the number of variables which are under consideration increases, the number of possible solutions also increases, but exponentially.

Figure 2.11 shows the result when a data set is projected into a space with too high dimensions. Both the figures show same data set, but the projected hyper

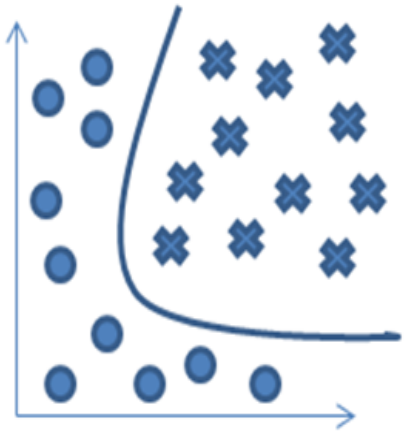


Figure 2.10: The good kernel function

plane comes from the SVM that uses a high-dimensional kernel function. The outcome is that the boundary between classes is very specific to the examples in the training set and hence over fitting occurs and it fails to generalize to new examples.

Therefore it is the largest practical difficulty which is encountered generally when applying an SVM classifier to a new dataset. One would like to use such a kernel function that will likely separate the data but without introducing too many irrelevant dimensions.

2.4.2 Reinforcement Learning (RL)

RL is based on the assumption of a stochastic environment without the possibility of knowing examples of best actions for specific situations. In RL, an agent builds a policy for solving a certain problem through a trial and error process, receiving feedback from the environment in the form of a reward associated with each tried

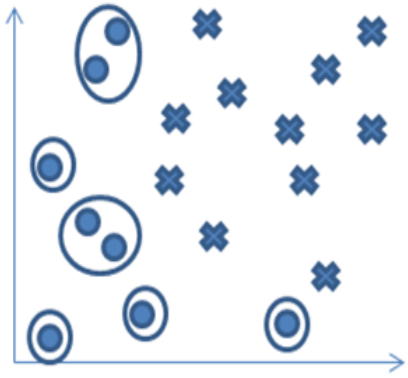


Figure 2.11: The kernel function with high values

action[136][44][31]. For agents interacting at discrete time steps t = 0; 1; 2; the basic reinforcement learning model consists of:

- Observes state $s_t \in S$
- Selects action $a_t \in A(s_t)$
- Obtains immediate reward $r_{t+1} \in R$
- Observes resulting state s_{t+1}

Figure 2.12 and Figure 2.13 summarizes the flow for RL.

Q-learning is a specific kind of reinforcement learning that assigns values to action-state pairs. In Q-learning for every state there are a number of possible actions that could be taken, each action within each state has a value according to how much or little rewards will be achieved for completing that action. First, consider the optimal Q-value function, the one that represents what's true of the

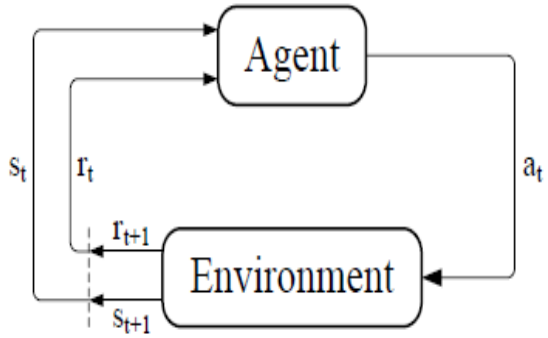


Figure 2.12: Reinforcement learning agent

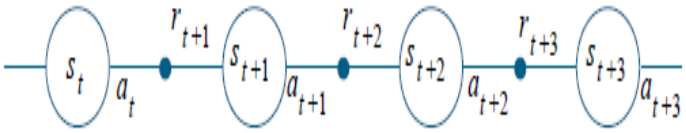


Figure 2.13: Flow for states and actions

world.

$$Q^*(x_t, u_t) = r(x_t, u_t) + \gamma max_{u_{t+1}} Q^*(x_{t+1}, u_{t+1}) \tag{2.1}$$

The optimal Q-value for a particular action in a particular state is the sum of the reinforcement received when that action is taken and the best Q-value for the state that is reached by taking that action. Agent processes this Q-value for each state and corresponding action. Initially it is any random value. As the learning happens to the system, this value keeps on optimizing.These values are saved in database.

Equation 1 is not the good way of calculating correct Q-value because in that there is an assumption that next state Q-value will be the best value. This is not

always true. This limitation is taken care in equation 2. A new learning rate, eta is introduced to control the learning step size. The new Q-value for the state and action is the weighted combination of the old Q-value for that state and action.

$$Q^{new}(x_t, u_t) = (1-\alpha)Q^{old}(x_t, u_t) + \alpha(r(x_t, u_t) + \gamma max_{u_{t+1}} Q^{old}(x_{t+1}, u_{t+1})) \quad (2.2)$$

For each failure, the Q-value for the node is reduced otherwise it keeps on increasing. This depends more on the reward values.

Self-healing techniques that use reinforcement learning in dealing with crashes and denial of service have been studied in [96]. In the case of a crash, the authors propose action selection techniques based on learning from previous experience. For self-healing, Reinforcement learning is chosen rather than continuing with SVM because of:

- correct input/output pairs are never available with SVM for different kind of failures.
- The RL agent can be modified (by adding punishments when the goals are not satisfied) to perform both reactive and deliberative decision making.
- RL can be very timeefficient, with algorithms for decision making performing in *O(m)* time for m possible actions on a learned policy.
- RL provides dynamic online learning, providing the ability of adapting to previously unseen situations and of managing uncertainty.
- Limited number of states and related actions are there when fault tolerance is considered. These can be easily managed using RL. RL is difficult to

manage where state action pairs are very large. For complex systems lot of processing is needed to first train the system.

Still as per [68] reinforcement learning is not suitable for non-stationary (dynamic) environments. In decentralised distributed systems, it is not possible for an agent to have perfect and complete knowledge of the state of its neighbors, connections and environment. Simplest way to handle this problem is to use the single-agent RL to be applied as multi-agent. In this, single agents will run on multiple nodes and the information gathered by each is combined to take a decision.

2.5 Problem formulation

Grid computing is a means to capture and gainfully harvest unused computer resources whereas autonomic computing sets out to simplify and automate overall computer operations more efficiently. They are separate, standalone initiatives developed to cope with different operational challenges. However, neither is fully fledged, as per its expected potential, as yet. For grid computing to succeed, it needs to be simple and automated to the point of being transparent. Ideally grid computing needs to be autonomic.

It became apparent that the modern grid systems have become more complex by their size and heterogeneity. Autonomic computing provides a new self* concepts to address all issues like self-configuration, self-healing, self-protection, and self-optimization. As per the literature reviewed, this is observed that many improvements can be done in the area of self-healing and self-protection of the grid systems. Self-healing will make the system capable to detect and recover from potential problems and continue to function smoothly. Many existing ap-

proaches as described in section 2.2.2 are there to heal the system from different kind of failures mentioned in section 2.2.1. A self-protecting system on other hand, is capable of detecting and protecting its resources from both internal and external attacks. Different kinds of network attacks like DoS, R2L and U2L are discussed in section 2.3.1 and existing work done is explained in section 2.3.2. Many drawbacks and the improvement areas are also mentioned as part of this literature survey.

Another weakness observed in current grid systems is their fragile nature mainly because of their static organization and the absence of alternate path of communications. Centralization introduces weakness in the system because of a single point of failure. They need to be sustained by the suitable overlays that are scalable and fault resistant. An agent-based system can be used to achieve this for grids.

In this work, our proposed model SHAPE achieves the above mentioned vision of an autonomic grid computing through multi-agent based approach. A multi-agent model has been developed that helps the grid to attain the self-healing and self-protection. The conventional multi-agent system is extended to support autonomic feature. The developed model helps to stabilize the fault tolerance and security mechanisms in current grid system. It will manage the intricacy, complexity and performance analysis in such dynamic system where nothing is predictable.

2.6 Objectives

The objectives of this research are:

i. To propose a self-healing and self-protecting model of agent-enabling autonomic computing for Grid environment.

ii. To design and implement the proposed model to handle different types of faults like hardware faults, software faults, network faults, etc and also to protect the system from the internal and external attacks.

iii. To test and validate the proposed model.

2.7 Summary

This chapter described the related research done in the area of self-healing and self-protection in grid computing. Self-healing refers to the ability to discover, diagnose, and recover from faults. Thus, the self-healing property enables a distributed computing system to be fault-tolerant by avoiding or minimizing the effects of execution failures. Common types of failures- hardware, network and software are discussed. Due to the complexity for dealing with these failures need for self-healing is identified.

Self-healing system is categorized into two categories: Fault Detection and Fault Healing. Fault detection is the process to monitor the system for failures and detecting any fault in a device. Fault healing provides the solution to recover from detected faults.

Self-protection refers to the ability to anticipate and protect against threats or intrusions. This property makes an autonomic system capable of detecting and protecting itself from malicious attacks so as to maintain overall system security and integrity. Self-protection system is presented as: System Level, Management

Level, and Related Technologies. During literature survey it is identified that, lot of work is done on the authentication and authorization of resources in grid but not much work is done in the area for protecting grids against various network attacks. The work on intrusion detection systems for grid is also discussed. All the existing research is based upon the static definitions of attacks inside the IDS. There is not much intelligence in the system to update itself against new attacks definition not present in the database.

Later part of the chapter discusses machine learning. Because of its self-learning capabilities machine learning algorithms are identified as the best fit for implementing self-healing and self-protection in complex systems like grids. Based upon the past surveys, State vector machine (SVM) for implementing self-protection and reinforcement learning for implementing self-healing are chosen. Chapter is concluded by defining the problem statement.

Chapter 3

SHAPE- Self Healing and Protection Environment

3.1 Introduction

Self-healing and protection environment (SHAPE) is the first combined self-healing and self-protection approach for complex distributed systems. SHAPE proposes a model to automatically diagnose problems from observed symptoms, and the results of the diagnosis can then be used to trigger automated response and recovery. This helps in dealing with an important problem of automation of distributed system management in aspect of recovery from different kinds of faults, as well as from a number of security attacks.

This chapter is organized as follows: Section 3.2 discusses the evolution of SHAPE. After getting aware of the iterations through which SHAPE came across, the design principles are defined in section 3.3. Detail architecture of SHAPE is discussed in section 3.4. SHAPE is the multiagent model, and agent communica-

tion details are shown in section 3.4.1. This section also includes agent's design, efficient communication and agent security, later in the chapter the individual components required for the working of SHAPE are discussed- Monitors (section 3.5.1), Analysis unit (section 3.5.2), Planner (section 3.5.3) and Executor(section 3.5.4). All these sections describe the detailed working of SHAPE.

3.2 Evolution of SHAPE

Work on SHAPE has been accomplished after going through various iterations (Figure 3.1).

First the need for automating grid computing has been identified after compar-

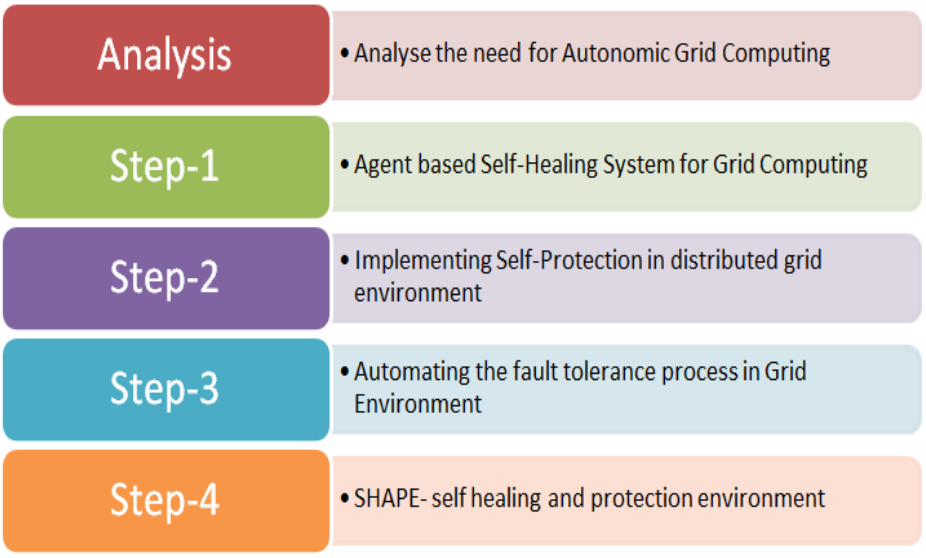


Figure 3.1: SHAPE evolution

ing current very prominent grid middlewares [52].Grid environment is a complex system where the deployment, management and maintenance is a tedious task. On the other hand, autonomic computing deal with the complex systems by in-

troducing the concept of self-management. Both grid computing and autonomic computing are promising computing paradigm. Following can be achieved by introducing autonomic features within the grid environment.

- To detect and identify hostile behaviour and then take automatic actions to protect the grid environment against intrusive behaviour.
- To dynamically tune system to meet end-user or business needs with minimal human intervention.
- To minimize all the outages in the system for keeping it up and available 24*7*365.
- To adapt automatically in the dynamically changing environments.

After analyzing the need for autonomic properties to help grid function properly, the two of the autonomic properties- self-healing and self-protection are identified. The first proposed model is a very basic model that extends the self-healing autonomic management unit (SMU)[53] to deal with failures. An agent-based Self-healing system (ABSS)[51] is proposed and implemented. This approach uses the concept of multi-agents and autonomic computing to provide an automated way to deal with failures.

ABSS (Figure 3.2) consists of three main components: Monitor Engine, Processing Unit and Fault Handler. ABSS interacts with meta-scheduler and resource registry information in a grid. Monitor engine picks the resource info from resource registry store within grid environment and starts monitoring the resources. It consists of different agents that perform different functions like Network, Application and Resource monitoring. Monitor passes all the information gathered

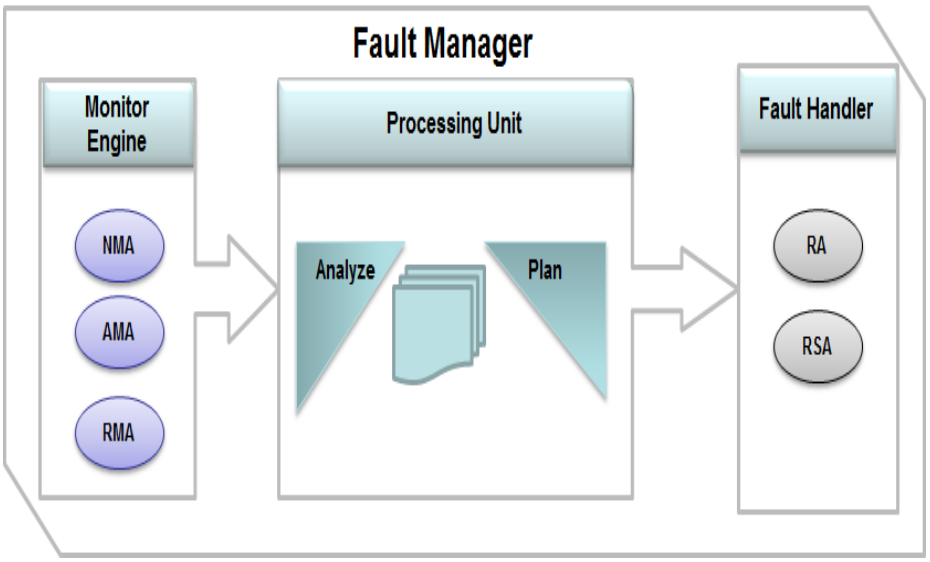


Figure 3.2: Agent Based Self-healing System (ABSS) model

about current grid status to processing unit. Processing Unit then analyze the information provided by the Monitoring Engine and pass this information to the fault handler. Fault handler takes the information and takes the proper action. Figure 3.3 show the complete setup view.

After having initial structure of the self-healing model i.e. ABSS in place, the next step is to build a model to provide self-protection to the system. Various approaches were inspected, and new model named Self-protection Model (SPM)[54] is evolved. SPM (Figure 3.4) is an agent enabling autonomic computing providing a promising solution to the system management troubles caused by increased complexity of large-scale distributed systems. SPM is based upon few implications of the genetic algorithm (GA). GA's are robust, inherently parallel, adaptable and suitable for dealing with the classification of rare classes. Moreover, due to its inherent parallelism, it offers the possibility to implement the system without requiring any additional resource. This new model adopts intelligent agents

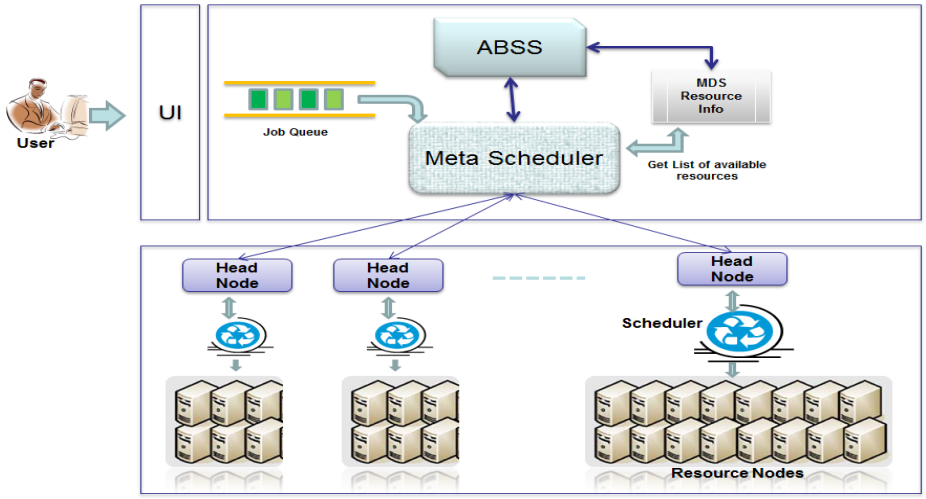


Figure 3.3: ABSS environment view

for dynamically organizing system management with centralized control. At the system level, each element contributes its capabilities on the functions of system management and cooperate with each other to implement autonomic computing for the grid system. Cooperative works are organized by dynamically associated relationships among autonomic elements (agents), including acquaintance, collaboration and notification. This self-organized model is more suitable for the grid environment as it is characterized by distributed, open and dynamic properties. Later few issues were faced while using the genetic-based approach. SPM was not scaling well for the systems like grids. As the number of resources increases, this also increases the mutation there, and this is like exponential increase in search space size. Another reason is the complexity of the implementation of the genetic approach within grid. It was observed that machine learning algorithms are better option to replace GA.

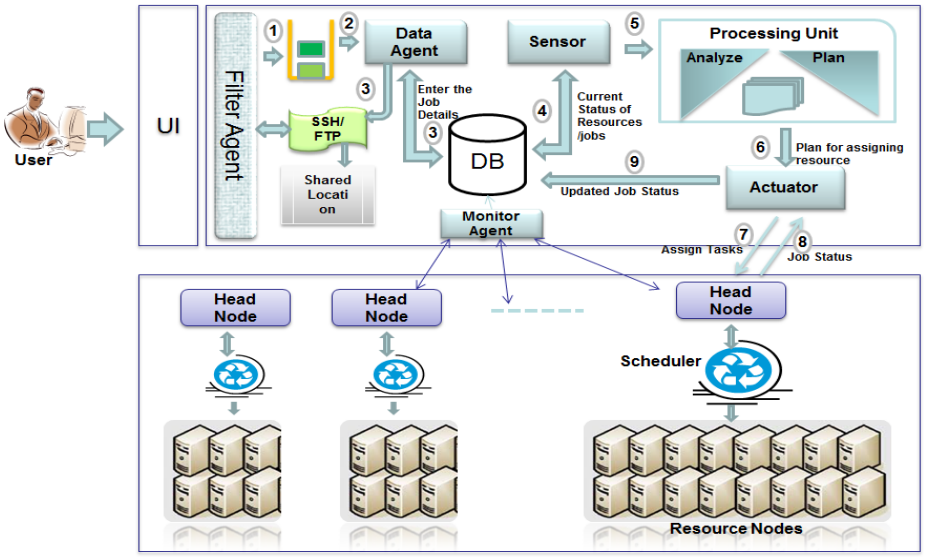


Figure 3.4: Self-protection model

After going through all these stages, self-healing and self-protection environment called SHAPE[55] is proposed. SHAPE offers an automated way of handling failures and provide protection from various kinds of security attacks. Self-healing will make the system detect and recover from potential problems and continue to function smoothly. A self-protecting system will be capable of detecting and protecting its resources from both internal and external attacks.

3.3 Design principles

A novel approach called SHAPE is designed that provides a platform for adding self-healing and self-protection capabilities to any distributed system. Architecture of SHAPE reuses the concept of autonomic computing to implement automated ways to deal with failures and security attacks. Complete implementation is done using open source technologies and it has component-based architecture

in which one can easily add or remove components.

Key design principles those are considered while designing SHAPE are:

For self-healing:

- Immediately determine the location of fault.
- Remove the faulty component from the rest of the network so that the remaining network works fine.
- Reconfigure or modify the network in such a way so as to have the minimal impact.
- Repair or replace the failed components to restore the network to its initial state.

For self-protection:

- A system needs to be able to detect intrusions with minimal false positives. It must be able to distinguish between valid and invalid data.
- The system must have the ability to respond to attacks.
- The components, involved in self-protection of the system, can become themselves a target of attacks. Those, if compromised, can be used by the attackers in an unintended way. Hence, the system must prevent self-protection components from being compromised.

3.4 SHAPE architecture

In this model, SHAPE autonomic elements (agents), which manage self-healing and self-protection of network, dynamically organize management tasks without

centralized control and directions. Autonomic element consists of sensors, monitors, analyzer, planner, executor and effector (Figure 3.5). The details of each component are discussed in sections given below. Each of SHAPE elements establishes an acquaintance relationship to acquire data from each other to keep them updated. Based on such acquaintance information, they are able to form collaborations by their interactions and contribute each member's capability to accomplish necessary sub-tasks when failure or some security breach occurs. Every participant acts according to its capability and knowledge, and send results, further request and information to others for cooperative work. With this architecture, SHAPE is also trying to be scalable, robust and reliable system for handling failures and attacks.

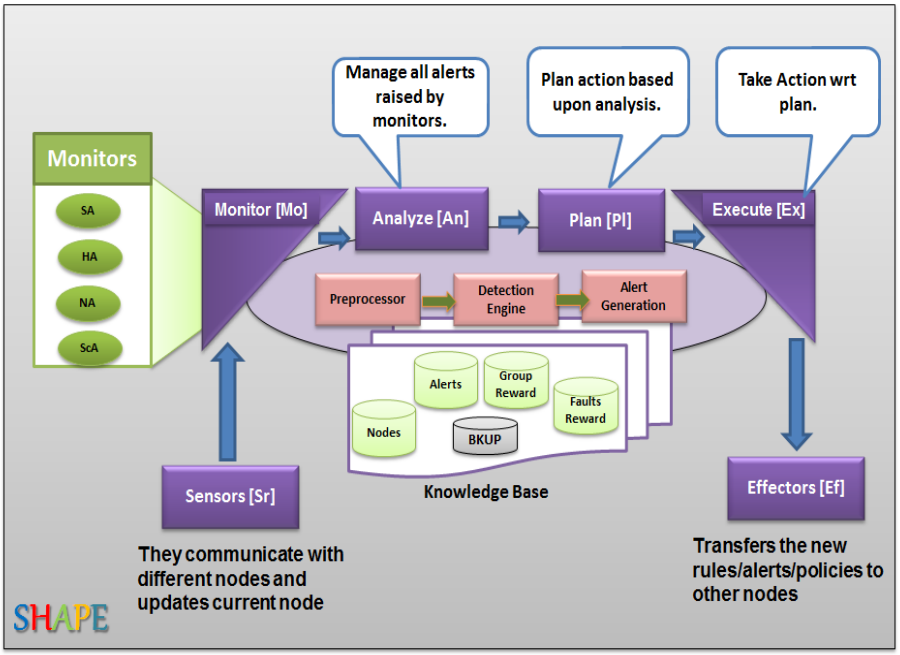


Figure 3.5: SHAPE autonomic element

Together when these Autonomic elements communicate, they form autonomic

unit (AU). AU consists of different machines (AE's) working together to handle failures and security breaches. One AU has single manager node and rest acts as the processing nodes to generate data for the manager. Only managers can communicate with the managers of another AU. To brief SHAPE components mathematically:

Autonomic Element, AE= {Sensor, Monitor, Analyze, Plan, Execute, Effector}

Autonomic Unit, AU= $\{AE_1, AE_2, ..., AE_N\}$

SHAPE= $\{AU_1, AU_2, ..., AU_N\}$

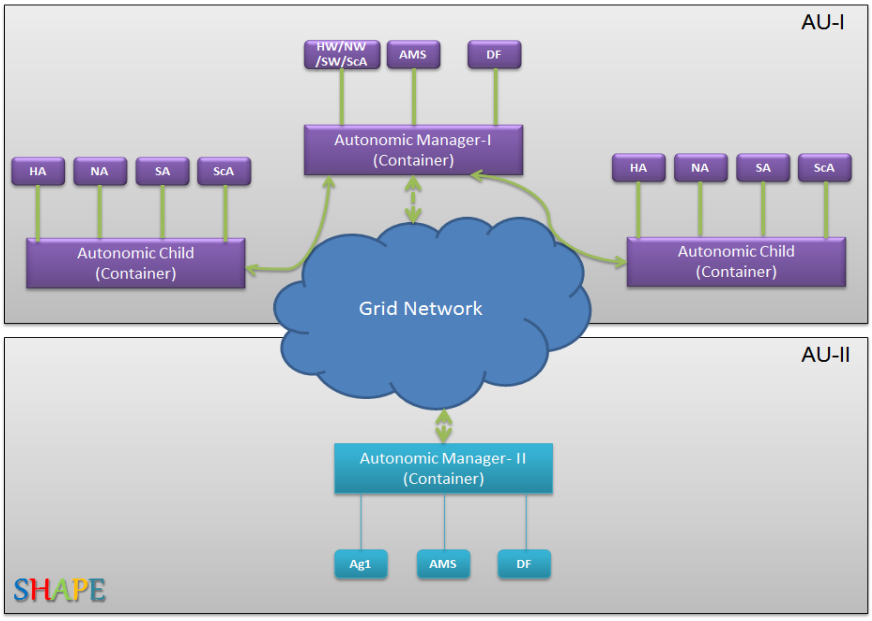


Figure 3.6: SHAPE autonomic unit interaction

3.4.1 Agents communication

This section discusses the communication details for agents. To have efficient communication, the discussion is done in three fold- Agent design, Efficient com-

munication and Agents security. Agents are designed to use JADE MTP and PKI modules for intelligent and secured communication.

3.4.1.1 Agent design

Java Agent Development Framework (JADE) [20] is used as a platform for creating and managing agents. Each agent is associated to a container that runs inside its JVM. In SHAPE, the autonomic manager will be regarded as the main agent and has few additional properties related to managing the child agents. Basic structure of agent-based communication is shown in the Figure 3.6.

Manager node is having two additional agents [43] Agent Management System (AMS) and Directory Facilitator (DF) running to manage the interaction between different AE's. The Agent Management System (AMS) provides supervisory control for accessing and using agent platform. Only one AMS will exist in a single AU. The AMS maintains a directory of agent identifiers (AID) which specifies agent state. Each agent must register with the AMS in order to get a valid AID. DF is an agent which provides yellow page service in the platform. All exchange of messages is controlled by JADE's message transport system also called Agent Communication Channel (ACC).

3.4.1.2 Efficient communication

JADE-MTP (message transport protocol) [75] is used to further optimize the communication between agents. This reuses connection instead of opening new ones each time a message must be delivered to a remote platform.

Communication between the manager and child node is both , push and pull based. As shown in Figure 3.7, AM (manager node) can ask from the status of AE

and if AE fails to do so for three times in sequence, it is treated as down.Whenever new updates are available, the manager node pushes them to child nodes and can pull the logs from the child nodes after specific interval.

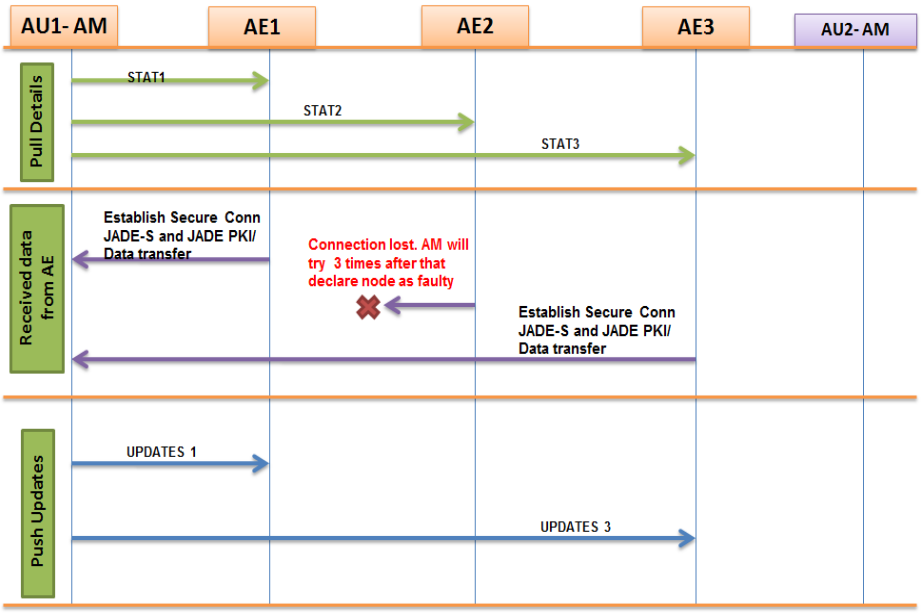


Figure 3.7: SHAPE communication

All the updates done by manager are stored in a centralized database. Manager node also maintains the DB replica of all configurations as backup. Whenever the master DB goes down due to any reason,backup DB acts as master till master DB is not up again. Also, by keeping the configurations in centralized DB any AE participating in that AU and depending upon its reward value (discussed later in chapter) can act as the manager node in case of failure of the manager.

3.4.1.3 Agents security

For security of agents, integration is done with JADE-PKI (public key infrastructure) [15]. JADE-PKI add-on introduces a public key infrastructure into JADE.

The add-on provides security for an agent messaging and secures the communications between the agents. Agent messaging is secured by the PKI Agent Messaging Service. It provides methods of encryption, signing, decryption and verification of the agents communication messages. An agent does not perform these operations itself, but gives it to the service. The agent only marks a message whether it should be signed and/or encrypted. [15] explains the steps to configure this module in JADE. Table 3.1 shows the security parameters those are configured for adding security to agents.

Table 3.1: Security parameters

Name	Description
jade_pki_keyStore	A path to a key store where a private key and a certificate of the container is placed.
jade_pki_keyStorePassword	A password to the store pointed with the jade_pki_keyStore parameter.
jade_pki_trustStore	A path to a store where certificates of trusted CAs are.
jade_pki_trustStorePassword	A password to the store pointed with the jade_pki_trustStore parameter.

3.5 Working of SHAPE

This section describes the detailed working of each component used to form the SHAPE. Each component is discussed in order in which it is getting used inside SHAPE.

3.5.1 Monitors

Monitors play an important role in the proposed model. Monitor performs two functions- helps in adding a new resource to correct group and to monitor all the resources for failures. Whenever a new resource is added into the environment, group id is assigned to the node based upon its configurations (Algorithm- 1). Resources with same configurations are placed in the same group and resources those are having new configurations are placed in a new group. These resources that belong to a new group are eligible for the hardening process. List of such nodes is passed to an autonomic unit to process driver hardening through executor.

Algorithm 1 Monitors- Add new node

```
1: BEGIN
2: G_N{G1, G2, G3,...,Gi} be the set of existing groups
3: Let N_C be current node added into the system
4: G_c = getGroup(N_C, configuration)
5: if G_C isNULL then
6:    create new group G_nw
7:    ADD N_c → G_nw
8:    Mark this node as eligible for driver hardening.
9:    ADD G_nw → G_N
10: end if
11: getGroup(Node N_C, Object configurations)
12: {
13: Let G be the complete set of groups
14: key = configurations.OS:configurations.CPU:configuration.MODEL

15: if G ∩ key != null then
16:    return SUCCESS
17: end if
18: }
```

For understanding new node addition process take as an example where G1 and G2 are the existing groups.

G1 configuration: $IBM, LinuxOS, corei7processor$

G2 configuration: $IBM, WINDOWS7, corei5processor$

Whenever nodes are added into the environment, they should be added into the correct group. Same devices should be marked in the same group.

G1=N1, N2, N3

G2=N4, N7, N8

One new node N5 is going to be added into the system. Since it is a new node, the new group labeled G_{nw} is created, and N5 is added into this group.

Gnw=N5

The information for this node is then passed further to autonomic unit for initiating driver hardening process.

The task monitors also check the nodes for three kinds of failures- Hardware, Software and Network. Respective monitor agents are configured to deal with these faults. Functioning of monitors has been automated by using reinforcement learning concept. Reinforcement learning is process to make system intelligently decide what action to take by using the state action concept so as to maximize reward. The system must discover which actions yield the most reward by trying different state action pairs. As discussed in chapter-2, Q-learning algorithm is used to add intelligence to the agents (Algorithm- 2).

In the case of success, reward value is increased by 0.01 and in case of failure, it is reduced by 0.1. Also, RL is used to maintain information about the best resource depending upon their history. SHAPE also handles an exception scenario in which the node stops working, and it still has a good reward associated wrt

Algorithm 2 Monitors- Update Q-value

1: Function **updateQvalue**(Reward r)
2: initialize Q(xt,ut)
3: xt initialize observe state
4: **for all** xt **do**
5: Choose $ut \in A(s)$ according to policy derived for Q
6: Take action ut and observe next state xt+1 and reward r
7: $Q^{new}(x_t, u_t) = (1 - \alpha)Q^{old}(x_t, u_t) + \alpha(r(x_t, u_t) + \gamma max_{u_{t+1}} Q^{old}(x_{t+1}, u_{t+1}))$
8: **end for**
9: }
10: EndFunction

its history. In that case, a separate list of such nodes is maintained. This list overwrites all the details gathered by RL until the node details are not deleted from this list.

3.5.1.1 Hardware agents (HA)

Hardware monitor agents (Algorithm 3) are used to monitor system for hardware failures and raise an alert if something wrong is detected. These agents monitor the "Machine Check" logs using MCELogs (Linux) and MCat (windows) utilities. Both these tools use good lexical analyser to read the logs and do an entry in the database. Any error detected by HA updates the Q-value based upon the reward value. The overall status of the node is decided based upon Q-value. The actual error alert is raised by them in the "Alert DB". Fields that are used from these logs to monitor faults include:

- Event Type- To know the severity of events raised. Only logs with event type- "ERROR" or "CRITICAL" are analysed by proposed model.

Algorithm 3 Hardware Agents (HA)

```
BEGIN
for all Groups G_c:G do
   Hd_N{N_1, N_2, N_3, ..., N_i} be the set of hardened Node in particular group
   for all Node Nc:Hd_N do
      PARSE MCE Logs
      GET EVENT_TYPE, EVENT_ID, SOURCE, LOG, TIME_STAMP
      if (EVENT_TYPE Equals (ERROR or CRITICAL) && LOG Equals HARDWARE Events) then
         UPDATE database with this LOG information along with respect to the group key and node name
         ALERT raise!
         set reward value as r=-0.1
         updateQvalue(r)
      else
         IGNORE
      end if
   end for
end for
```

- Event ID- Uniquely identify the event.
- Source- It is the software that logged the event, which can be either a program name, such as "SQL Server", or a component of the system or of a large program, such as a driver name. For example, "Elnkii" indicates an EtherLink II driver.
- Log- The name of log file where the event was recorded.
- Time Stamp- Time at which the error event was raised

Figure 3.8 shows an example that how the reward value is updated for a particular node. With the change in reward new Q-value is re-calculated. This

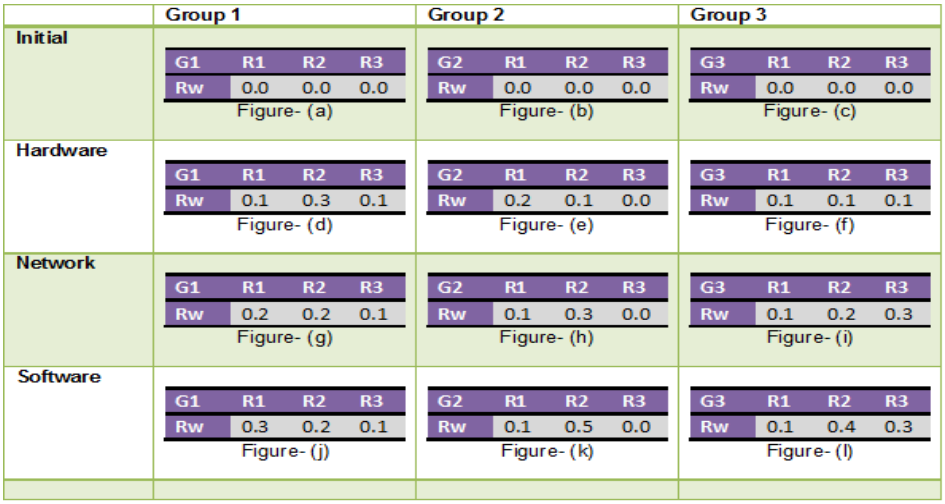


Figure 3.8: Q-value for different monitors

Q-value is for node level. Figure 3.8d shows the initial Q-values for different nodes within the group. These values are updated based upon the above analysis. Figure 3.8e and 3.8f shows that the reward values for different resource nodes based upon the error/success encountered.

3.5.1.2 Network agents (NA)

NA's (Algorithm 4) are used to detect network related failures and log the details into the "Fault DB'. Pull based heart beat model to update the node status to the manager in that AU. In pull-based model, the master node periodically asks for the child nodes status to check their existence. In the absence of response for continuous three times, the master considers some failure has occurred at that node. The status update period (P1) value is kept quite small for the nodes participating in the job execution. For the ideal nodes during that period are placed in the second segment with period (P2). Frequency of P2 in getting status update

is more than P1. This helps in reducing the overhead of default monitoring Figure 3.8(g-i) shows the example values gathered based upon the network failures. These values are used to select the best node in terms of network.

Algorithm 4 Network agents (NA)

```
1: BEGIN
2: int TIMER_SHORT=5Sec
3: int TIMER_LONG=90Sec
4: while true do
5:   for all Groups G_c:G do
6:     List IDEAL_NODES N_id = {N_1, N_2, ..., N_n}
7:     List WORKING_NODES N_wr = {N'_1, N'_2, ..., N'_n}
8:     for all IDEAL_NODES do
9:       Get node Status
10:      Thread.sleep(TIMER_LONG)
11:    end for
12:    for all WORKING_NODES do
13:      Get node Status
14:      Thread.sleep(TIMER_SHORT)
15:    end for
16:    if STATUSisnull then
17:      if !N_cinIGNORE_LIST then
18:        Add N_c to ignore list
19:      end if
20:    else
21:      if N_cinIGNORE_LIST then
22:        Remove N_c to ignore list
23:      end if
24:    end if
25:  end for
26: end while
```

3.5.1.3 Software agents (SA)

Software failures are continued to be the major concern in system reliability. Software monitor agents helps to monitor:

- Memory and CPU: Updates the CPU and Memory usage of the particular nodes involved in job execution. A cron/at job task keeps on running at the background that monitors the node CPU and memory usage. If the memory consumption or CPU usage is more than the threshold value, then an alert is raised. This threshold value is auto configurable and value is set after analysing the average load on environment by the manager node.

- Application: SA's constantly monitors the processes previously configured. List of applications to be monitored is configured within the agent. When agent detects that any process has stopped working it will raise an alert and add it in "IGNORE_LIST". Another scenario monitored by application monitor is when application is partially working i.e it is throwing exceptions while executing. "ALERT DB' is updated in both scenario.

3.5.1.4 Security agents (ScA)

ScA's (Algorithm 5) are used to check the system for various known and unknown attacks. ScA captures all the new anomalies and logs the details into the database. Table-3.2 shows the attacks from which SHAPE targets to protect the system.

SHAPE is using snort [134] as an anomaly detector to self-protect the system from security attacks. Snort has been optimized to be integrated with SHAPE. Security agents run on each node participating in the grid and logs the details in a database on Manager node of that AU [Table-3.2]. This is done using preprocessor that reads the log and represents each data instance as a vector of real numbers. This preprocessing is mandatory for the working of the detection engine that is used. The next function of the security agent is to raise an alert. It reads the

Table 3.2: List of attacks for which SHAPE deals

Attack Class	Attack Name	Description
DOS	1. Smurf 2. Neptune 3. Land 4. Teardrop	Attacks that disrupts a host or a network service in order to make legitimate users not to use that network.
R2L (Remote to Local)	5. Guess Password 6. IMAP 7. SPY	Unauthorized attacks gain local access from a remote machine and then exploit that network.
U2R (User to Root)	8. Buffer Overflow 9. Rootkits	Local users get root access without authorization and then exploit the network.
Probing	10. NMAP 11. Ports Sweep	Attackers use programs to automatically scan the network for gathering information or finding known vulnerabilities.

predicted patterns of packets reaching network ("Network Profile") and compare them with the packets captured earlier. It then logs an alert when the current value exceeds 'minimum' to 'maximum' range for that time.

SHAPE uses the concept of Support Vector Machine (SVM) to act as network profile. SVM on basis of training data interprets data and recognizes the patterns among them. Security agents work in the following manner:

Numerical conversion The necessity of this numerical conversion is underlined by the fact that only the numerical data can be used as an input to the SVM. The numerical conversion involves two sub-categories of tasks. The first

Algorithm 5 Security Agents (SA)

```
1: Input: Training data is already configured.
2: BEGIN
3: Capture packets using libPCap.
4: Parse captured packets.
5: for all Parsed Packets do
6:     Mapping symbolic features to numeric value.
7:     Scaling attribute data to fall within the range [-1, 1].
8:     Apply RBF kernel with best C and gama values.
9:     compare(training data, testing data)
10:    update training data based upon above method.
11: end for
```

one is used for training the SVM. This serves as the final training data which is responsible for the learnability of the SVM. The second one is used for the testing of SVM. This implies that this data is tested based on what SVM has learnt. Table 3.3 shows the different labels and their associated snort rule fields. The steps for numerical conversion include:

- Read Snort rules file.
- Remove all the "alert" strings.
- For external and home network use the values without dot delimiter and place additional zeros to complete three digit set.
- For protocols replace TCP with 01, UDP with 02, ICMP with 03 and IP with 04.
- For message ASCII code of each alphabet is taken and then addition of all the codes serves as the numerical equivalent for the message field

Table 3.3: Snort rule option and their SVM label

Label	Keyword	Description
1	Protocol	TCP, UDP, ICMP, and IP
2	IP Addresses	Source and destination IP
3	Port Numbers	Port on which request is made
4	msg	The msg keyword tells the logging and alerting engine the message to print with the packet dump or alert.
5	reference	The reference keyword allows rules to include references to external attack identification systems.
6	sid	The sid keyword is used to uniquely identify Snort rules.
7	rev	The rev keyword is used to uniquely identify revisions of Snort rules.
8	classtype	The classtype keyword is used to categorize a rule as detecting an attack that is part of a more general type of attack class.
9	priority	The priority keyword assigns a severity level to rules.
10	metadata	The metadata keyword allows a rule writer to embed additional information about the rule, typically in a key-value format.

Continued on next page

Table 3.3 – *Continued from previous page*

Label	Keyword	Description
11	content	The content keyword allows the user to set rules that search for specific content in the packet payload and trigger response based on that data.
12	rawbytes	The rawbytes keyword allows rules to look at the raw packet data, ignoring any decoding that was done by preprocessors.
13	depth	The depth keyword allows the rule writer to specify how far into a packet Snort should search for the specified pattern.
14	offset	The offset keyword allows the rule writer to specify where to start searching for a pattern within a packet.
15	uricontent	The uricontent keyword in the Snort rule language searches the normalized request URI field.
16	pcre	The pcre keyword allows rules to be written using perl compatible regular expressions.
17	ttl	The ttl keyword is used to check the IP time-to-live value.
18	id	The id keyword is used to check the IP ID field for a specific value.

Continued on next page

Table 3.3 – *Continued from previous page*

Label	Keyword	Description
19	dsize	The dsize keyword is used to test the packet payload size.
20	flags	The flags keyword is used to check if specific TCP flag bits are present.
21	flow	The flow keyword allows rules to only apply to certain directions of the traffic flow.
22	seq	The seq keyword is used to check for a specific TCP sequence number.
23	ack	The ack keyword is used to check for a specific TCP acknowledge number.
24	window	The window keyword is used to check for a specific TCP window size.
25	icmp_id	The icmp_id keyword is used to check for a specific ICMP ID value.
26	icmp_seq	The icmp_seq keyword is used to check for a specific ICMP sequence value.
27	rpc	The rpc keyword is used to check for a RPC application, version, and procedure numbers in SUNRPC CALL requests.
28	ip_proto	The ip_proto keyword allows checks against the IP protocol header.

Training data Training data is used to train an SVM model file that has the capability to determine whether that pattern corresponds to an attack or some normal pattern. In order to train the SVM initially, the predefined snort rules are converted to a numeric format as discussed above and are used as the training data.

- A parser is written that reads the alert file and then convert it into the numeric format. The converted output is then saved into the file with same name as that of the snort rule file with different extension as .nrules.
- Now, this file is used to enhance the learnability of SVM by defining the patterns and expected outputs corresponding to it. In future, whenever the SVM encounters the exact pattern match or some similar pattern then it has the ability to distinguish it into attack or normal data by looking into this file.

Testing data This is used as the testing data for the SVM. Any new entry through preprocessor is tested in order to determine whether it is an attack or not. This is done by looking for a match if available with the SVM training data.

- Any data from the preprocessor for testing the SVM is firstly converted into a numerical form. This conversion is performed in a manner similar to the one mentioned above.
- If there is match of data, code will detect the matching percentage of the string. In case of all the strings with matching percentage less than 90% and greater than 70%, a new entry is made in the .nrules file and alert is raised.

- Accuracy of the SVM can also be predicted by analyzing the difference between expected and actual output on the basis of the training data available to the SVM. This is done by AU.

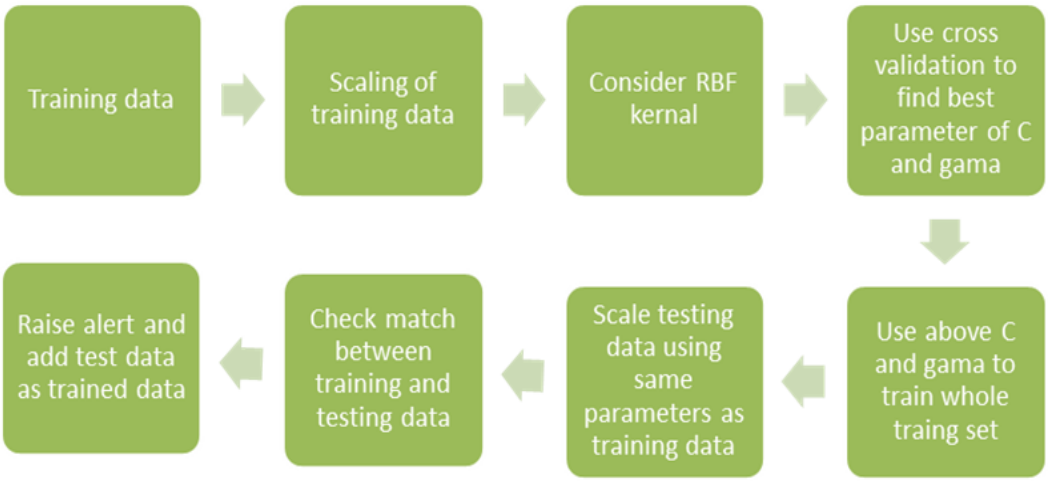


Figure 3.9: Working of security agent

Implementation of SVM Figure 3.9 shows the complete working of the security agents. First step is to get the training data by going through the above mentioned steps. The scaling of this input data is done to ensure the dominance of attributes in smaller numeric ranges. Moreover, it also avoids the numerical difficulties arising during the calculations because of large attribute values. Each feature is scaled in the range [-1, +1] or [0, 1]. The same method is used to scale both training data, as well as testing data. Out of the four commonly used kernels, radial basis function (RBF) kernel is used as it can handle the case when the relation between class labels and attributes is nonlinear. Additionally, it has fewer numerical difficulties. In order to increase the accuracy with which the classifier foretells the output for unknown testing data, the best possible values

are selected for the parameters (C, gamma) by using cross-validation. In v-fold cross-validation, the training set is first divided into v subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining v - 1 subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data correctly classified. Now the values of parameters obtained are used to train the model on the training set. As a result, a model file is obtained which is used for the classification of testing data. The parameters used to scale the training data are saved and retrieved when scaling is performed on the testing data. After the successful completion of data scaling, testing is done on the scaled data. The output file is then obtained which contains the predicted labels for the testing data.

For example- consider a string input to the SVM. Firstly, it is parsed. Afterwards, that parsed string is converted to a numeric form by applying logic above discussed. This is done inside preprocessor. Now the training set available with the SVM also consists of numeric data in the same format as this input string which is already converted to a real number format but also with an output label appended to each string. The choice is made for the closest match between the input to SVM and the data in the training set. The output is predicted on the basis of this closest match.

Below string shows one sample numeric format training data string:

1 1:01 2:172003016033 3:567 4:6789435578 6:289 7:1 11:98446883577 19:3

Training set contains the following strings:

1 1:01 2:172003016033 3:567 4:6784324421 6:289 7:1 11:65764323577 19:3

1 1:01 2:172003016035 3:568 4:6789435578 6:289 7:1 11:67343222214 19:2

1 1:01 2:172003016039 3:569 4:6789435578 6:289 7:1 11:63436823222 19:2

1 1:01 2:172003016053 3:567 4:6789435578 6:289 7:1 11:98446883344 19:3

Now the training set in SVM however doesn't contain an exact match, still it foretells the output on the basis of closest match, i.e. the output will be the one for which maximum number of characteristics match. Hence, here the output predicted by the SVM is 1.

In addition, C and gamma values directly influence the accuracy of the SVM. A smaller value of C allows ignoring points close to the boundary, increasing the margin, and risk of underfitting. When C is large, we increase the variance (try to fit as close as possible to the training data) with the risk of overfitting. For small values of gamma, the decision boundary is nearly linear. As gamma increases the flexibility of the decision boundary increases. Large value of gamma leads to overfitting.

3.5.2 Analysis Unit

Data related to failures logged by monitors are processed in AU. AU helps to further refine the data gathered and to figure out the correct action that can be taken based upon specific type of failure. This information is used to update the "Faults reward" table. Working of AU has been divided into following steps:

Defining system states: The first step is to identify system states in which failure can occur. More are the number of states identified, more is the learning time. Choosing the correct number of sates is important. They should be small enough to cover all the fault scenarios.Reducing states count will speed up the

decision process. Table-3.4 shows the states identified for this system to be optimized and cover most of the scenarios.

States 1-3 represents network failures. During heartbeat status check, when node is not reachable, these type of faults are considered under PingIP, faults that are more due to DNS not reachable or pinging from node is not possible can become part of state 3 and 2. State 4-6 categorize hardware failures. "System running status" depends upon the hardware responding status. "Driver check" list down all the failures related to hardware drivers because of hardening. Any hardware failure can be covered under hardware component state. State 7-9 are related to software related failures. Applications that are consuming extra memory or cpu are covered under state-7. State-9 picks those applications that are not running smoothly. These also include applications that are causing latencies in completing the jobs. Exceptions captured from the logs can be processed through state-9.

Defining actions Each state has one or more associated actions. The number

Table 3.4: Fault States

Index	States
1	Ping IP
2	Ping Gateway
3	DNS Lookup
4	System Running Status
5	Driver Check
6	Hardware Component Failure
7	Memory/CPU
8	Application not Responding
9	Exceptions in execution

of available actions at each state will depend on the number of properties that can be modified to correct that failures associated with the state. Table-3.5 shows

the actions identified.

Action A-C helps to correct network related failures. Action D-F list the techniques to resolve hardware failures and remaining actions G-I are there for software failures.

Calculating rewards Rewards are calculated irrespective of the groups. Based

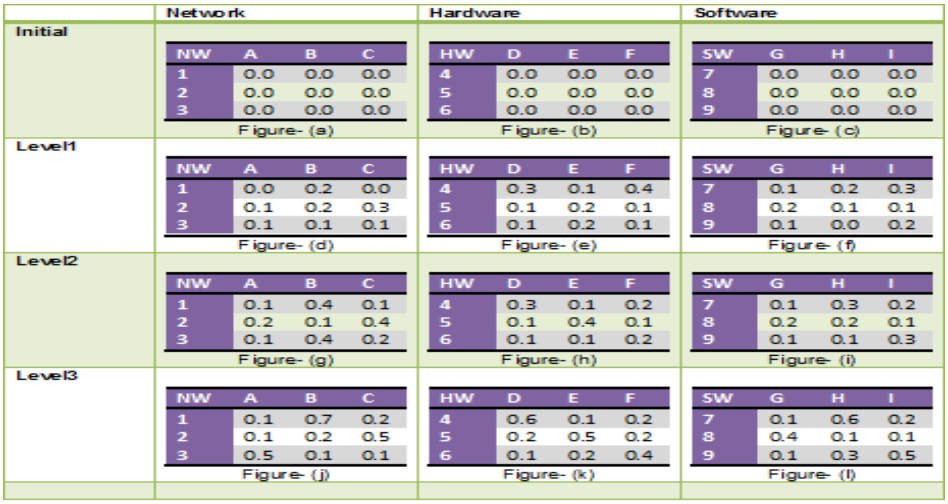
Table 3.5: Actions associated to state

Index	Action
A	NW Restart
B	Renew IP
C	Block Node- Alert NW
D	HW Restart
E	Revert Drivers
F	Block Node- Alert HW
G	SW Restart
H	Memory Cleanup
I	Block Node- Alert SW

upon the failures that are in particular state and the action decided to correct those failures, reward calculations are done. If the action taken helps in recovering the state, then reward value is increased for that particular state action pair.

All the alerts- hardware, network and software logged during the monitor phase will be analysed by the analysing unit. Figure-3.10.a,b,c shows that initially all the reward values for different states are zero. During the training phase, actions are chosen randomly for the failures in particular state. If that action corrects the failure, the reward value for that action is increased. In this way the best possible action can be chosen for some particular state.

Once analysis of logs is done, action is taken to reduce the failure rate for



	Network	Hardware	Software
Initial	NW A B C 1 0.0 0.0 0.0 2 0.0 0.0 0.0 3 0.0 0.0 0.0 Figure- (a)	HW D E F 4 0.0 0.0 0.0 5 0.0 0.0 0.0 6 0.0 0.0 0.0 Figure- (b)	SW G H I 7 0.0 0.0 0.0 8 0.0 0.0 0.0 9 0.0 0.0 0.0 Figure- (c)
Level1	NW A B C 1 0.0 0.2 0.0 2 0.1 0.2 0.3 3 0.1 0.1 0.1 Figure- (d)	HW D E F 4 0.3 0.1 0.4 5 0.1 0.2 0.1 6 0.1 0.2 0.1 Figure- (e)	SW G H I 7 0.1 0.2 0.3 8 0.2 0.1 0.1 9 0.1 0.0 0.2 Figure- (f)
Level2	NW A B C 1 0.1 0.4 0.1 2 0.2 0.1 0.4 3 0.1 0.4 0.2 Figure- (g)	HW D E F 4 0.3 0.1 0.2 5 0.1 0.4 0.1 6 0.1 0.1 0.2 Figure- (h)	SW G H I 7 0.1 0.3 0.2 8 0.2 0.2 0.1 9 0.1 0.1 0.3 Figure- (i)
Level3	NW A B C 1 0.1 0.7 0.2 2 0.1 0.2 0.5 3 0.5 0.1 0.1 Figure- (j)	HW D E F 4 0.6 0.1 0.2 5 0.2 0.5 0.2 6 0.1 0.2 0.4 Figure- (k)	SW G H I 7 0.1 0.6 0.2 8 0.4 0.1 0.1 9 0.1 0.3 0.5 Figure- (l)

Figure 3.10: Q-value for different monitors

job execution. AU keeps check that new job submission should not be done to already faulty nodes, save the current state of the job and then restart the node, raise an alert if the restart still not resolves the issue.

Security alert logs: contains the security breach alerts raised by different security agents. AU update the planner about these alerts and publish SID associated with the alerts.

3.5.3 Planner

Planner plans the action based upon the information gathered during the monitoring and analysis phase. The "Group reward" table prepared during the monitoring phase is used for selecting the best nodes available for the job execution. "Faults reward" table generated by the analyser based upon the monitor agent information is used to take the action to deal with the failures.

Each number in "Group reward" and the "Faults reward" table will be the

Algorithm 6 Planner

```
1: if ResourceSelection request then
2:     var resourceID = getResourceID(groupID)
3:     return resourceID
4: end if
5: var action = getActionForState(state)
6: pass action info to executor
7: executor will then execute the action to deal with failure
8: FUNCTION getActionForState(state)
9: Select action from FaultReward where MAX(reward)
10: ENDFUNCTION
11: FUNCTION getResourceID(state)
12: Select resourceID from GroupReward where MAX(reward)
13: ENDFUNCTION
```

latest estimate of the probability for taking the respective action. This estimate is treated as the state's value, and the whole table is the learned value function. Say state A has a higher value than state B, or is considered "better" than state B, if the current estimate of the probability of our winning from A is higher than it is from B.

In the Figure 3.10 considering group G1, resource R1 has the highest Q value. This is considered as the best node to start the job execution with. Similarly for other groups G2 and G3, resources R1 and R3 are the best options.

3.5.4 Executor

Executor (Algorithm 7) main role is to perform the actions that are decided by the complete process. Main features of executor includes: driver hardening, resource selection, failure handling and update anomalies definition to different

agents participating in the network.

Driver hardening HHA targets to reduce the failure rate because of any hardware cut downs. Whenever the node is added into the network, this agent checks for its device drivers and tries to harden them. Device driver acts as an interface between hardware and the application in any system. The device and driver interact through a protocol specified by the hardware. When the device obeys the specification, a driver may trust any inputs it receives [10].Device hardware failures cause system hangs or crashes when drivers cannot detect or tolerate the failure.

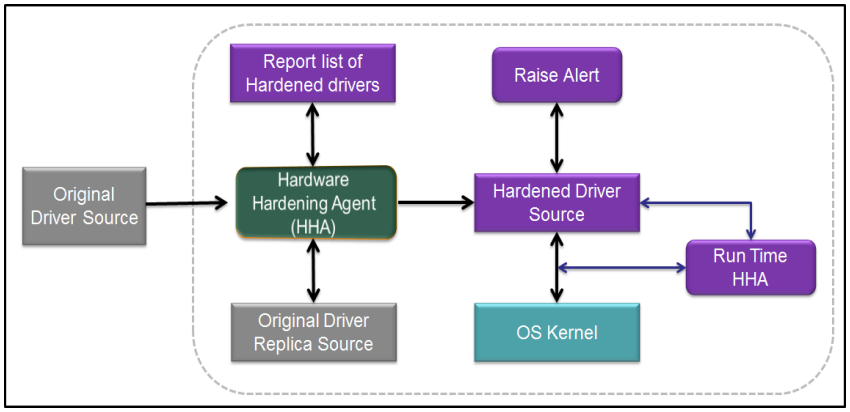


Figure 3.11: Working of hardware hardening agent

SHAPE uses the concept of carburizer [10] to harden the device drivers. Hardening is the process in which driver works correctly even though faults occur in the device that it controls or other faults originating outside the device. A hardened driver should not hang the system, or allow the uncontrolled spread of corrupted data as the result of any such faults. Our implementation is different from carburizer in the way that our system is based upon the agent-based architecture.

Algorithm 7 Executor

```
BEGIN
FUNCTION hardwareHardening(){
G_N{G1, G2, G3,...,Gi} be the set of hardened Node
Let N_C be current node added into the system
G_c = getGroup(N_C, configuration)
if G_C isNULL then
    create new group G_nw
    ADD N_c → G_nw
    START HHA hardening process
    Scan drivers and generate list of drivers to be hardened, L_i
    Create replica of original drivers that are going to be hardened
    for all L_i do
        Harden driver using carburizer engine
    end for
    ADD G_nw → G_N
else
    Hardened driver already available.
    UPDATE N_C
end if
getGroup(Node N_C, Object configurations)
{
Let G be the complete set of groups
key = configurations.OS:configurations.CPU:configuration.MODEL

if G ∩ key != null then
    return SUCCESS
end if
}
} ENDFUNCTION
FUNCTION resourceSelection(){
Pass the best node information Group, ResourceId to scheduler
passed information will be used for job execution
} ENDFUNCTION
FUNCTION failureHandling(){
Take the respective action passed by planner
if  SUCCESS then
    reward = reward + 0.1
else
    reward = reward - 0.01
end if
} ENDFUNCTION
```

Secondly, HHA generates reports listing all hardened device drivers and updates the manager node. CIL (C Intermediate Language) is used to design this component. CIL using its tool set and high representation level, helps in analyzing C programs and their source-to-source transformation.

Whenever a node is registered, SHAPE driver hardening agent pushes the code on that node. The complete command is executed through SSH session. Once the hardening process is over, HHA updates nodes status to manager. Figure 3.11 shows the working of HHA. HHA generates replica of original drivers and log detail for the drivers to be hardened, whenever it is going to harden those drivers. Harden process includes first scanning the source code of all the drivers and find out the code where the chances of failure are. Once the code is identified, it starts replacing the code so as to harden the driver. After the driver is hardened, original drivers are replaced with the hardened drivers. A runtime monitoring HHA component is also added to every hardened driver. This monitoring component keeps track over the proper functioning of the driver. If any alerts are raised because of the misbehavior of driver, hardened driver is replaced with the original driver and the manager node is updated. In such scenarios, manual intervention of the programmer is needed.Once the problem is resolved, the alert raised is manually resolved by the programmer and hardened driver is again updated. The fix related information is kept in the database for future reference.

Chances that fault remains unnoticed are nearly negligible as proper logs are inserted into driver during hardening process. This helps to track the exact behavior.

Resource selection and failure handling Planner passes the node information and the action to be taken for correcting failures to executor. Executor then based upon the information provided take the required action.

3.6 Summary

This chapter discussed self-healing and protection environment called SHAPE. After presenting the evolution of SHAPE model, the detailed architecture of SHAPE is discussed. SHAPE is an agent-based approach, which is built upon the autonomic computing architecture as a base. Agent framework is implemented using JADE. The communication between the agents is done with the help of JADE MTP and the communication is secured by integrating JADE PKI module. It involves various autonomic elements communicating with each other to self-protect the system from attacks and failures. The autonomic element consists of sensors, monitors, analyzer, planner, executor, and effector.

Monitors register new nodes with respect to their configuration and monitor the system for faults and security breaches. Monitors are categorized into different categories- hardware, network, software and security. Hardware, network and software agents perform their functions by using reinforcement learning algorithm while security agents being more dynamic in nature use SVM for keeping check over security attacks. Analyzer analyzes the information gathered by the monitors and pass related information to planner. Based upon the input from analyzer, planner plans proper action and pass action information to executor to take proper action. Executor also does hardening of device drivers using modified version of carburizer. Next chapter will describe implementation and results.

Chapter 4

Experimental Details and Results

4.1 Introduction

This chapter discusses the process of realizing SHAPE in practice according to the proposed work. For implementing it grid environment is setup using Globus (Figure 4.1). After having the grid environment in place, SHAPE is implemented over the grid environment to provide self-healing and self-protection capabilities.

Section 4.2 discusses the details about the environment and the technologies used to implement SHAPE. Overall validation of SHAPE is divided in two categories- self-healing and self-protection system validation (section 4.3). Different standard metrics are used for validation. Section 4.3.1 and section 4.3.2 reports and discusses results.

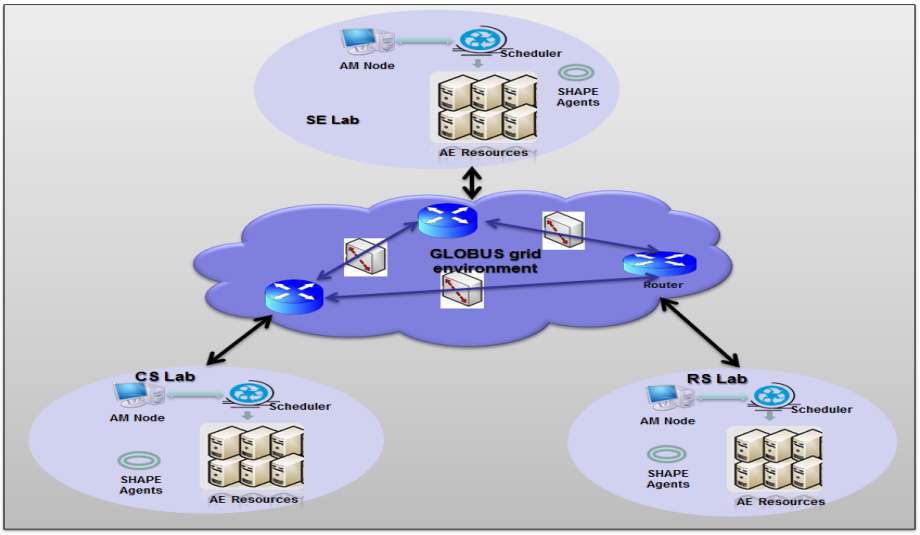


Figure 4.1: Environment overview

4.2 SHAPE Experimental Setup

Proposed model is tested on the Grid environment at Thapar University (TU), setup using Globus toolkit. Globus toolkit 4.0 has been installed on different resources and the service they provide were deployed as grid services. Along with Globus, proposed model agents are also placed on each node to automatically handle faults and security attacks. Grid environment setup consisted of 44 Intel Dual Core 2.2GHz processor Windows XP nodes, 20 dual 2.4GHz Xeon Linux nodes and 5 nodes 450MHz PII Linux nodes. Each node has 1GB RAM and 80GB HDD [Table- 4.1]. TU Grid is exposed to the outer world with limited access. These systems are further categorized in three AU's for experimentation purpose, though there is no hard rule to limit autonomic elements (AE's) in an autonomic unit (AU).

Various packages used for implementing SHAPE are:

Table 4.1: SHAPE environment details

Configuration	OS	Number	AU
Intel Dual Core 2.2GHz	Windows	44	AU1,AU2
2.4GHz Xeon	Linux	20	AU3
450MHz PII	Linux	5	AU3

- Grid environment setup using Globus 4.0

  Over GT4.0 grid environment SHAPE implementation was done. SHAPE provides an additional security and fault tolerance features as discussed in Chapter 3.

- Oracle Java SDK 6.0

  Java is used as development language to implement SHAPE components.

- JADE SDK

  JADE is used for managing agents in secured and controlled way. Various plugins like JADE-PKI and JADE-MTP are used along with default JADE SDK for effective and secure communication.

- LIBSVM SDK

  LibSVM[26][30] library is used to implement SVM. It provides a simple interface where users can easily link it with their own programs. The main features include:

  - Different SVM formulations
  - Efficient multi-class classification
  - Cross validation for model selection
  - Probability estimates

  - Various kernels (including precomputed kernel matrix)
  - Weighted SVM for unbalanced data
  - Both C++ and Java sources
  - GUI demonstrating SVM classification and regression

- Eclipse IDE for JAVA

  Eclipse is an integrated development environment (IDE). It helps in making the implementation easy and in more manageable manner.

- Snort AD 3.0

  Snort as an anomaly detector is taken and modified to act as the intelligent system to deal with various network attacks.

- SPSS

  Statistical analysis of various self-healing systems with SHAPE and default grid environment was tested using ANOVA test using SPSS 20.0 statistical package (SPSS Inc Chicago, Illnosis, U.S.A.)

4.3 Results and Discussions

SHAPE validation is done in two folds-self-healing and self-protection.The details for each are discussed below. With the help of experiments performed, it is proved SHAPE to be scalable, robust and reliable system for dealing with failures and security attacks. To check the scalability, varying number of jobs are submitted to grid and SHAPE handles a growing amount of work in a capable manner. Also, being the multi-agent based architecture, starting from quite a few

resources participating in a grid during unit testing, resources are increased to three different autonomic units with different configurations. With the increases in resources, the system is responding without any delays. It is also proven with help of standard metrics that SHAPE is able to handle simulated failures and security attacks without any failure. This tests the robustness and reliability of the system.

4.3.1 Self-healing validation

For verifying self-healing, different experiments are conducted with variations in number of jobs submitted and percentage of faults injected. Based upon these experiments, average throughput, average turnaround time, average waiting time and failure rate are calculated. Standard Metrics for verifying SHAPE self-healing feature includes (Table- 4.2):

Table 4.2: Self-healing metric

Self-Healing Metric	Description
Throughput	Number of jobs executed in given time.
Turnaround Time	Interval between job submission and job execution.
Waiting Time	Amount of time for which job has to wait before its execution starts
Failure Rate	Percentage of failures detected by the system.

Throughput is defined as the number of jobs executed in given amount of time. Throughput is one of the most important standard metrics used to measure performance of fault tolerance system.

Throughput(n)=n/T_n

Where 'n' represents total number of jobs

'T_n' is total amount of time necessary to complete 'n' jobs.

In general the average throughput of all techniques decreases with increase in the percentage of faults injected and with the increase in the number of jobs submitted to the system. Table 4.3 shows reported values. Mean and standard deviation values were calculated for different data sets of faults. p-value is obtained to check the consistency between the results obtained for SHAPE and default globus based grid environment. Figure 4.2 and Figure 4.3, shows the graphical representation of mean and standard deviation, with and without SHAPE agents in place. When SHAPE's Self-healing module is used to handle faults, the number of jobs executed per hour increases then the normal scenario. Figure 4.2, shows the throughput calculated when 1000 jobs with varying faults percentage were executed. With the increase in faults, mean throughput decrease but remains always more than the system without any self-healing in use. As the number of jobs increases, mean value of throughput using SHAPE keeps on increasing wrt the GT4 based grid system. The p-value 0.041 also proves the effectiveness of result.

Turnaround time From particular job point of view, the important criterion is how long the system takes to execute that job. The interval from the time of submission of a job to the time of completion is the turnaround time.

$$T_a(n) = \sum_{k=1}^{n} (Tc_k - Ts_k)/n$$

Where for n number of jobs,

Ta= Turnaround time

Table 4.3: Throughput based comparison

Metric/ Fault Percentage		Throughput (N=1000)		Throughput (N=5000)	
		SHAPE	Default	SHAPE	Default
10%	Mean	16.51	16.01	5.77	4.27
	Std. deviation	5.01	4.67	1.45	1.11
	p- value	0.067		0.061	
20%	Mean	16.05	14.39	6.2	2.33
	Std. deviation	3.17	3.45	1.12	0.55
	p- value	0.061		0.058	
30%	Mean	19.23	11.87	9.82	1.59
	Std. deviation	3.56	4.21	1.67	0.42
	p- value	0.059		0.05	
40%	Mean	21.77	6.73	11.29	0.69
	Std. deviation	2.1	2.75	.73	0.18
	p- value	0.057		0.041	

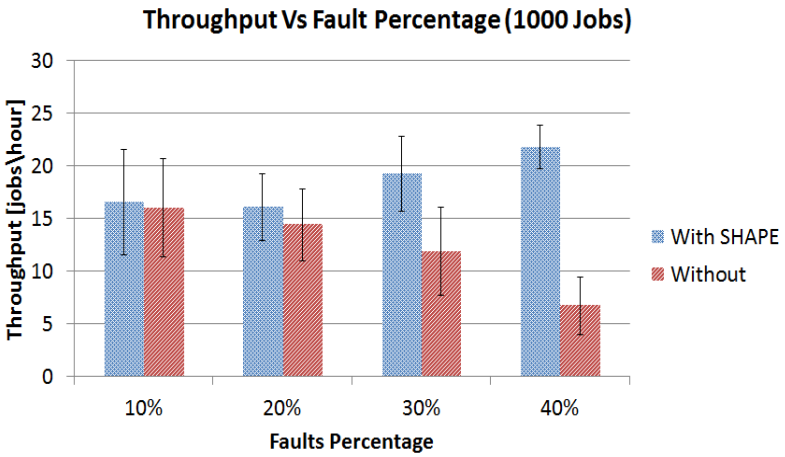


Figure 4.2: Throughput vs Fault Percentage (1000 Jobs)

Tc= Completion Time

Ts= Submission Time

Figure 4.4 and Figure 4.5 graphically shows turnaround time for the SHAPE

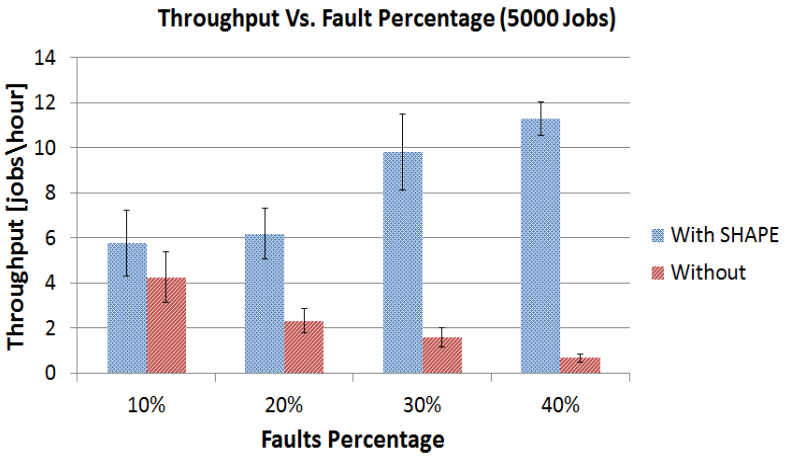


Figure 4.3: Throughput vs Fault Percentage (5000 Jobs)

system without any self-healing system deployed. System is tested for 1000 and 5000 jobs with different fault percentage. Table 4.4 shows mean and standard deviation values obtained by conducting experiments. It is found that with increase in number of failures in system, 'Ta' increases. As compared to normal scenarios, in SHAPE 'Ta' is less, because in SHAPE all failures are handled automatically and this increases the speed of execution.

Another reason is the good selection of nodes when job execution is about to start. This is based on the information gathered about each node by different SHAPE agents. This information is then further analysed by the analysing unit of SHAPE and the status of each resource is updated in the central database. Whenever new job is going to start, the resource allocation is done wrt current status of nodes available in database.

Table 4.4: Turn around time based comparison

Metric / Fault Percentage		Turnaround time (N=1000)		Turnaround time (N=5000)	
		SHAPE	Default	SHAPE	Default
10%	Mean	213.11	229.45	1192	2000
	Std. deviation	80.11	83.25	123.1	195.89
	p- value	0.067		0.00	
20%	Mean	219.71	321.95	1331	3289
	Std. deviation	87.71	121.65	121.01	229.23
	p- value	0.059		0.01	
30%	Mean	318	685.26	1422	4010
	Std. deviation	102.49	200.24	110.13	245.98
	p- value	0.052		0.031	
40%	Mean	332.7	1389.72	2139	5210
	Std. deviation	121.03	224.65	167.77	239.01
	p- value	0.032		0.027	

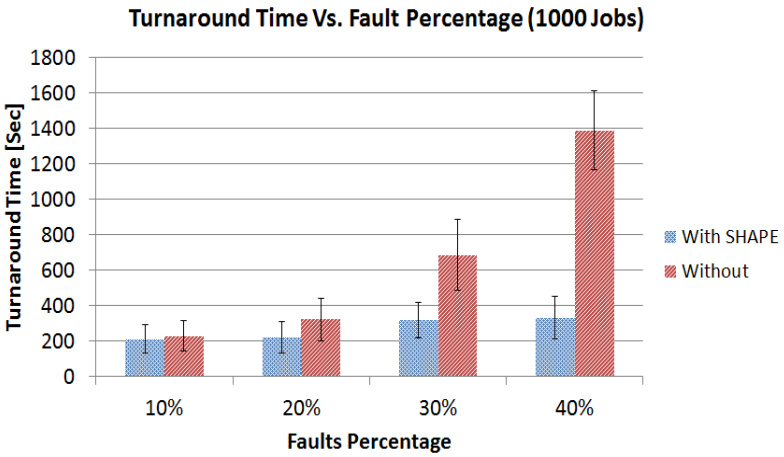


Figure 4.4: Turnaround Time vs Fault Percentage (1000 Jobs)

Unavailability/waiting time Waiting time in simple words is the delay in start of job execution. It is the average amount of time between submitting the job to the system and starting the execution on one of the execution nodes.

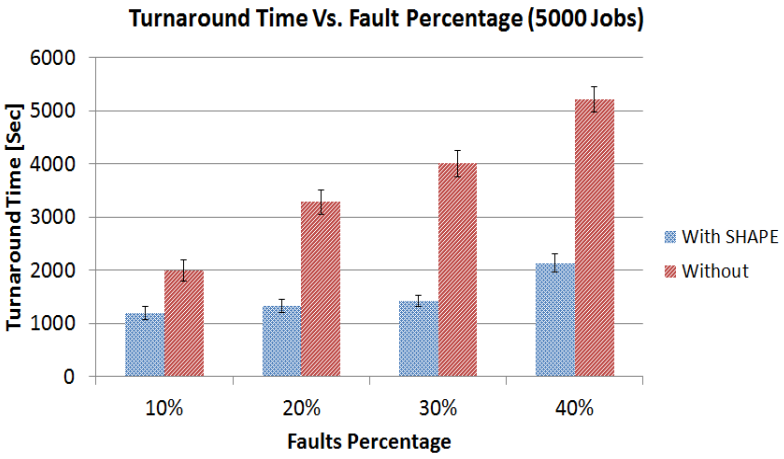


Figure 4.5: Turnaround Time vs Fault Percentage (5000 Jobs)

$W_t(n) = \sum_{k=1}^{n}(Te_k - Ts_k)/n$ Where for n number of jobs,
Wt= Waiting Time

Te= Execution starts time.

Ts= Submission Time

Figure 4.6 and Figure 4.7 shows Wt for job to start execution with SHAPE agents deployed and without SHAPE agents. With different percentage of faults injected in system, 'Wt' keeps on increasing with their increase numbers. SHAPE based model has 'Wt' less than the normal system. The reason is that SHAPE has pre-planned 'Node' categorization wrt the current state of nodes. This decreases the execution start time 'Te' and thus decreases waiting time. Detailed results can be referred in Table 4.5.

Failure rate Failure rate represents the percentage of failures detected by the system. Figure 4.8 shows the failure tendency of SHAPE. SHAPE works for

Table 4.5: Waiting time based comparison

Metric/ Fault Percentage		Waiting time (N=1000)		Waiting time (N=5000)	
		SHAPE	Default	SHAPE	Default
10%	Mean	179.1	230.1	556.14	1230
	Std. deviation	23.41	29.7	27.66	33.12
	p- value	0.067		0.04	
20%	Mean	201.37	254.31	735.23	1640
	Std. deviation	26.88	22.41	33.12	61.98
	p- value	0.069		0.038	
30%	Mean	178.12	397.11	813.71	2003.21
	Std. deviation	33.29	40.81	41.98	86.39
	p- value	0.057		0.021	
40%	Mean	227.19	798.93	1020.18	2673.25
	Std. deviation	37.21	38.66	48.23	107.2
	p- value	0.051		0.033	

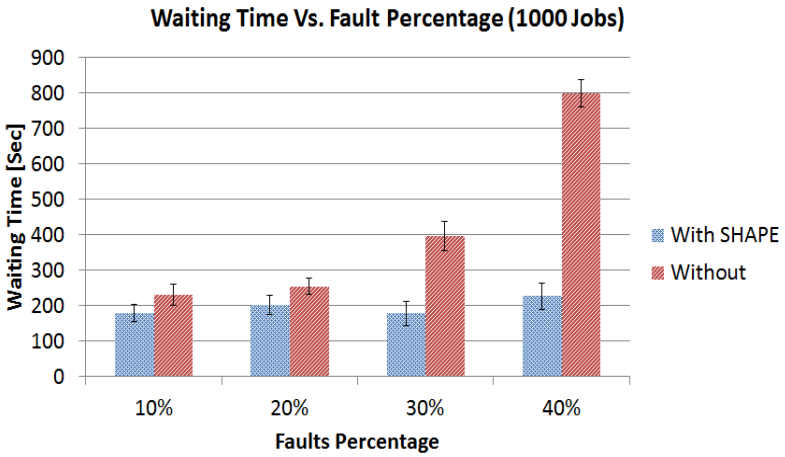


Figure 4.6: Waiting Time vs Fault Percentage (1000 Jobs)

three kinds of failures- Software, Network and Hardware. Tests are performed with constant number of failures introduced for different number of jobs. Based upon the tests performed, it is shown that SHAPE keeps on detecting different

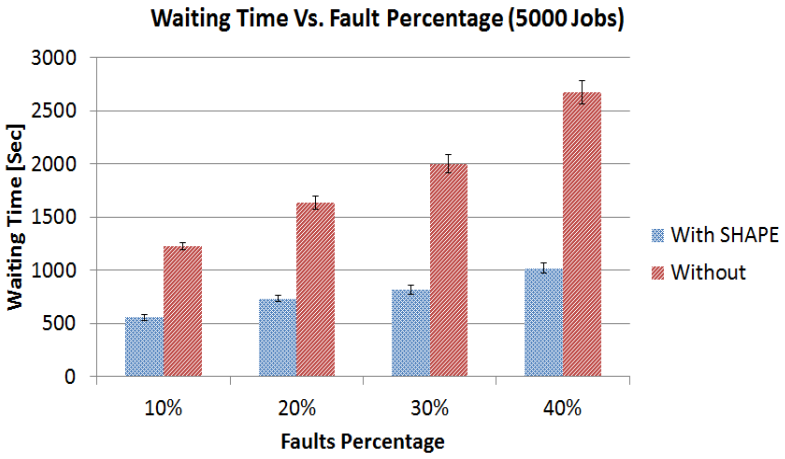


Figure 4.7: Waiting Time vs Fault Percentage (5000 Jobs)

types of failures irrespective of number of jobs submitted. The best fault detection rate is for hardware failures.

$F_t = \sum FailureType/Totalnumberof failures$ Where, Failure Type will be (Hw = Hardware Failures, Sw =Software Failures and Nw =Network Failures)

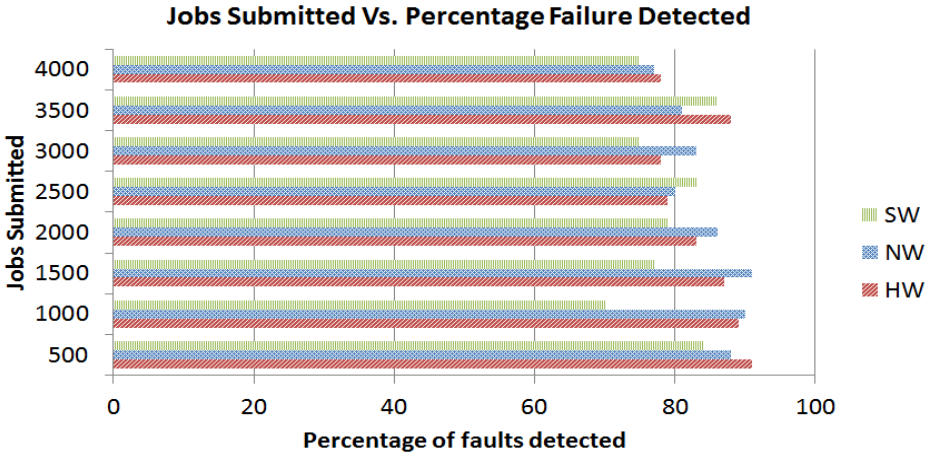


Figure 4.8: Jobs Submitted vs Percentage Failure Detected

Reward values Figure 4.9 shows the reward values wrt different fail states and the actions taken for network, hardware and software failures. For better clarity of results, the reward values based upon their actual values are reduced to the scale of 5. Network failures show that in case of failures where the monitor doesn't receive a heart beat response, the best way is to release and renew the IP for that particular node. These types of failures might be due to the sudden conflict of static IP's of nodes or due to network not responding. Second category of network failures taken into consideration is gateway is down which is causing the resources to break from the actual network. Here network restart can be a good option to get the network back. For network failures related to DNS errors, alert should be raised, and entry for that node should be placed into ignore list.

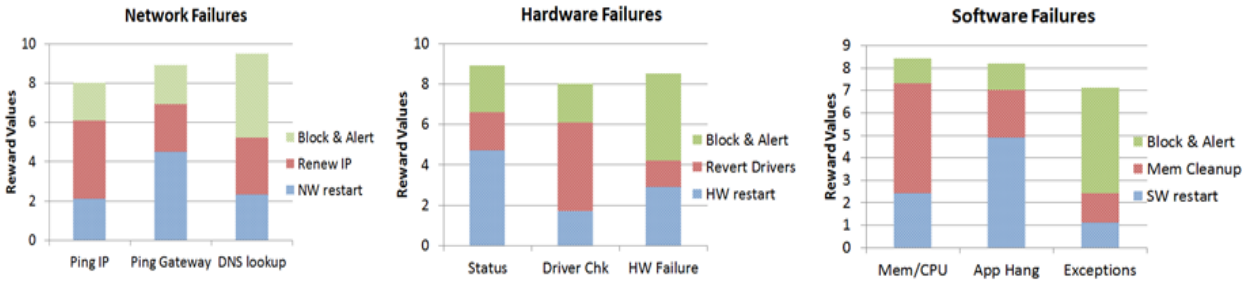


Figure 4.9: Reward values based upon state action

Hardware failures show that if the monitor agent fails to obtain the status of hardware, the best action identified based upon the history of reward values is to restart the hardware. Still after restart if the hardware is found faulty, alert is raised, and that node entry is made into ignore list for not using that node until repaired. If monitor agent found problem with the hardened drivers after analyzing logs, the best solution is to revert the driver to its initial version.

Software failures are mostly encountered due to lack of memory/cpu availability. In that cases, the best reward value is found to be of memory cleanup. In scenarios where application hangs, the best solution is to restart application and if still this doesn't resolve the problem, alert is raised for that application.

Figure 4.10 shows the reward values for different resources within different groups. The resource reward values helps to select the best resource. Resources with higher values are better option for executing critical tasks.

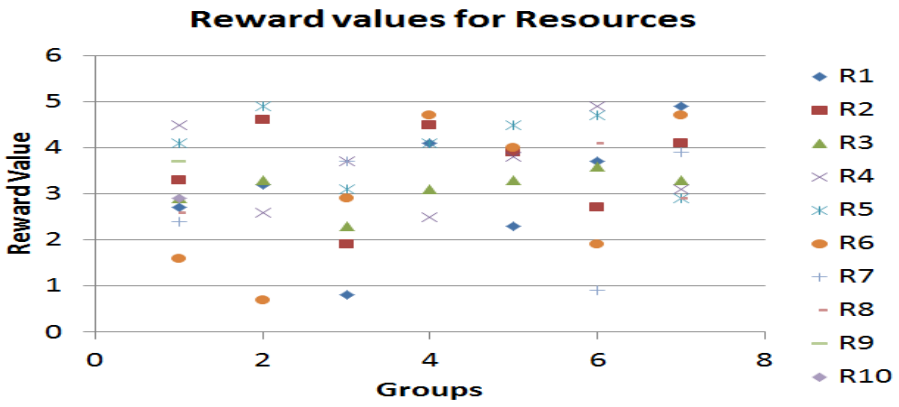


Figure 4.10: Reward values for resources in groups

4.3.2 Self-protection validation

In order to explicate how attackers can take the advantage of security vulnerabilities in computer systems or software services, in each attack category, attacks are simulated using different tools. For DDoS attacks, metasploit framework is used to launch smurf, land, neptune and teardrop attacks. For "remote to local(R2L)", hydra is used to do guess password and spying. NetCat is used to simulate "local to root" attacks. NMap is used for network probing. To evaluate the efficiency of

the SHAPE Self-Protection capabilities, following metrics are used (Table- 4.6).

Table 4.6: Self-protection metric

Self-Protection Metric	Description
Detection Rate	Number of attacks detected and blocked.
False Positive Rate	Total number of normal instances that were incorrectly considered as attacks.
Accuracy	Percentage of successful detection.

Detection rate Detection rate DR, is the number of attacks detected and blocked.

DR= Total No. of True Positives/Total No. of intrusions

SHAPE's detection rate keeps on increasing with time. As the system crosses the training period, its capability to detect new attacks keeps on increasing. For every new attacks/intrusions detected, the signature database keeps on updating with the new rules to prevent the same attack/intrusion from reoccurring. When more known attacks are there, then the detection rate in both cases is nearly same.

From Figure 4.11, it is clear that when attacks occur on the network are already known, and their signature exist in all the three systems, then the detection rate of all the approaches is nearly same. In Figure 4.12, when more unknown attacks are fired on the network, SHAPE is quite effective than even snort as anomaly detector. For verifying this, few known attacks signatures are removed and then launch those attacks on network.

When more unknown attacks are added into the system, then the snort as signature based system fail to detect unknown attacks. Detection rate for SHAPE

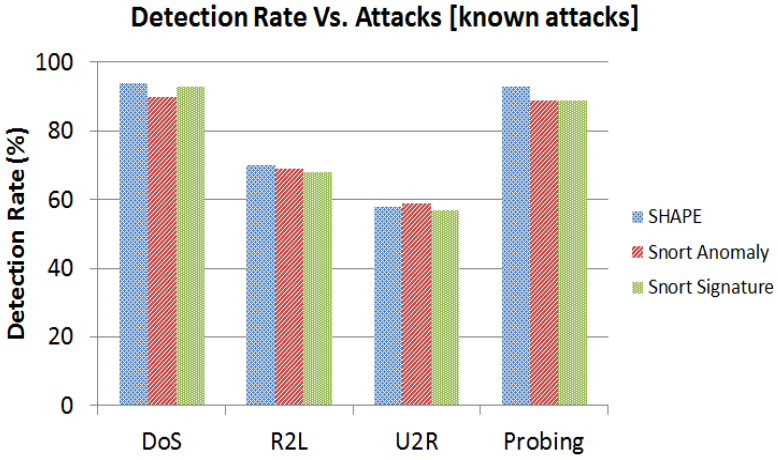


Figure 4.11: Detection Rate vs Attacks [known attacks]

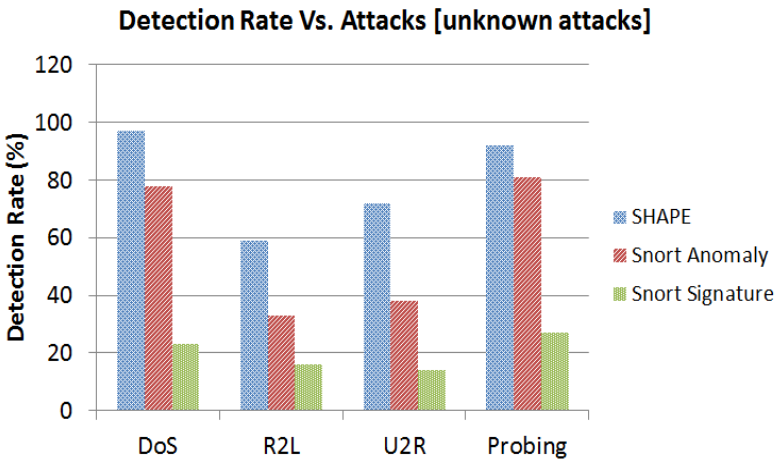


Figure 4.12: Detection Rate vs Attacks [unknown attacks]

keeps on increasing with time (Figure 4.13). Initially, when SHAPE is deployed, detection rate is less as the learning is minimum. As time passed, the detection capabilities of SHAPE increases. To verify this, the combination of known and unknown attacks in packets passed to security agent.

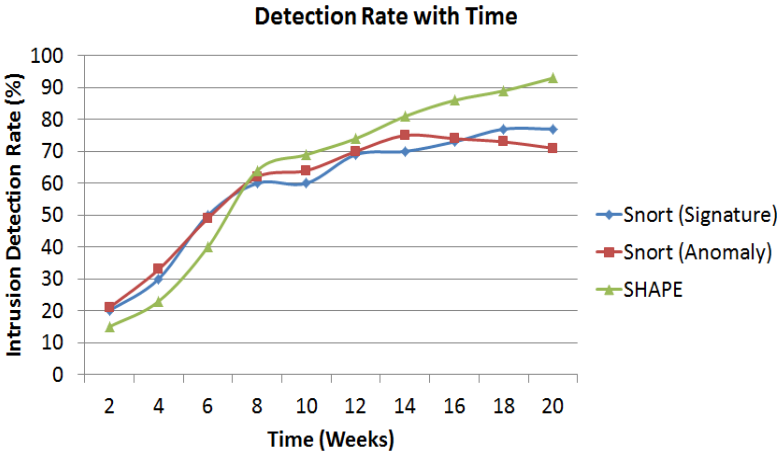


Figure 4.13: Detection Rate vs Time

False positive rate False positive rate is the total number of normal instances that were incorrectly considered as attacks.

Mathematically, False Positive Rate is $FP_R = FP/(TN + FP)$

where FP= False Positives and TN= True Negatives.

In SHAPE the FP_R keeps on decreasing with the passage of time. Figure 4.14, shows different categories of attacks and their false positive rates. With time, network profile managed by SHAPE become more stable, and this reduces the false positive rate. For DDoS attacks, FP_R is little higher as there are quite large number of variances in these attacks as compared to other attacks.

Accuracy Mathematically, Accuracy $A = TP + TN/(TP + TN) + (FP + FN)$

Accuracy helps to understand the success of the system in dealing with different attacks. Figure 4.15 and 4.16 shows the accuracy of the SHAPE in comparison to snort as anomaly and signature based system. From the graph it is clearly visible that accuracy of the new approach is better than the old snort based detection

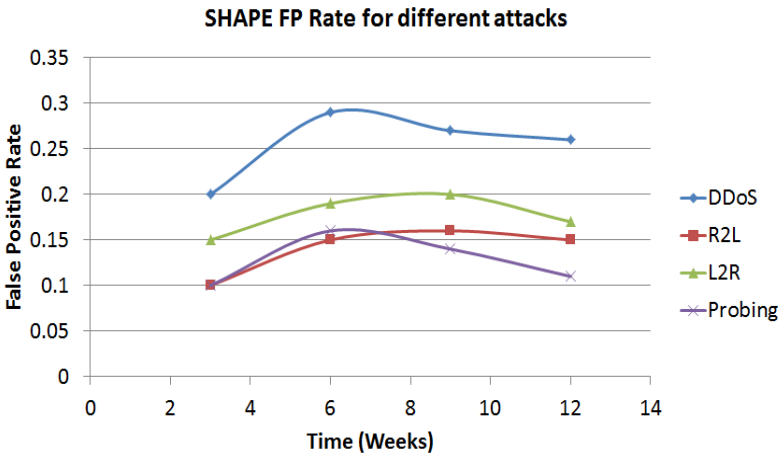


Figure 4.14: False Positive Rate vs Time

system.

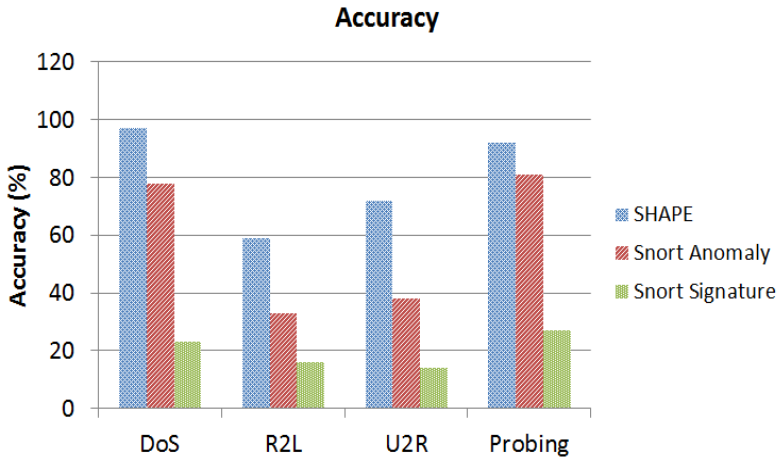


Figure 4.15: Accuracy for detecting unknown attacks

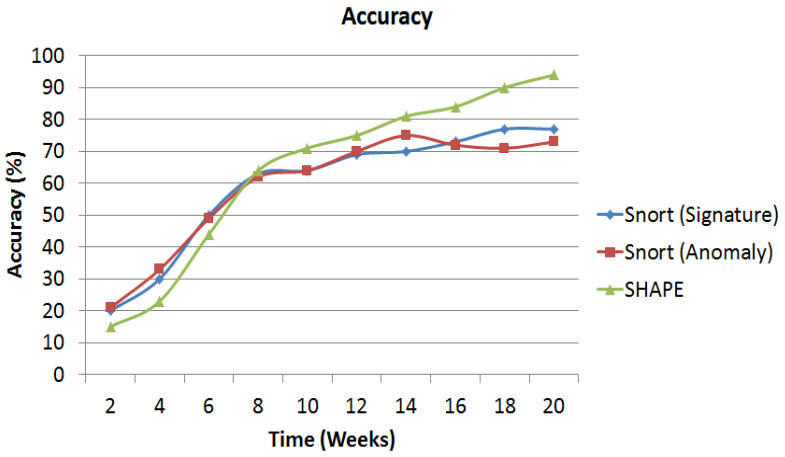


Figure 4.16: Accuracy with respect to time

4.4 Summary

This chapter dealt with experiments and results part of the thesis. After listing the basic requirements for SHAPE implementation we have proposed the system that deals with the problems discussed in Chapter 2. The system is implemented using JAVA as programming language, and few other tools are modified and integrated to develop a system called SHAPE. The experimental evaluation for our implemented system has been done by considering different test cases.

SHAPE is implemented purely using open source technologies, and results are provided to support the research. From results, it is proved that SHAPE reduces the failure rate and also increases the efficiency of system by executing a higher number of jobs per hour. It is also capable of detecting new anomalies and then automatically generating policies to block their reoccurrences.

Chapter 5

Conclusions and Future Work

This chapter summarizes what has been detailed in previous chapters. It also identifies few enhancements that can become part of future directions. Proposed autonomic model in this thesis helps in implementing self-healing and self-protection in grid computing using multi-agents. Model effectively overcomes the limitations of current fault tolerance and security approaches. Contributions and findings of this thesis are summarized in section 5.1. Section 5.2 presents the future aspects related to the research done. This section is further categorized into three subsections- Section 5.2.1 presents future work that can be done in terms of self-healing. Section 5.2.2 lists down the areas which can further improve the self-protection capabilities. Last section 5.2.3 describes the future of SHAPE with reference to cloud computing.

5.1 Conclusions

As the complexity of distributed system increase, they become more unreliable due to the involvement of largely heterogeneous resources located in different geographical domains. For these types of system, secure and fault-tolerant resource allocation services have to be provided. This thesis work presented a model for

effectively dealing with different kind of failures and various network attacks in Grids.

Following are the chapter wise key findings:

In Chapter 1, introduction to Grid computing and different kinds of grids is discussed. Due to scale of complexity and heterogeneous nature of the grid as compared to traditional computing systems, existing fault tolerance techniques and security against attacks of traditional systems are not enough to manage the faults in grid computing. This chapter also portrays the journey of grid computing to clouds and how the problems of fault tolerance and security are common to both the systems. Next in the chapter the title of the thesis is discussed that includes autonomic computing, self-healing, self-protection and multi-agents.

In Chapter 2, a taxonomy of self-healing and self-protection in Grids is provided. A detailed survey of various existing fault tolerance and security techniques are discussed to demonstrate the comprehensiveness of the taxonomy. The chapter starts with a brief introduction and then describes self-healing and self-protection in detail. Various critical failures and security attacks are listed, and the current approaches around these are mentioned. From the taxonomy and survey, the need for automating fault tolerance and security is identified. Machine learning based approaches are later discussed to add self-learning capabilities to the system. From the reported research already in the field of machine learning, SVM and RL based Q-based machine learning algorithms are the best algorithms with respect to the current problem of self-healing and self-protection. The problem statement is then formulated to conclude the chapter.

Chapter 3 presented Self Healing and Protection Environment (SHAPE) to automate fault tolerance and security from attacks in Grid environments. SHAPE is

the first initiative to provide combined self-healing and self-protection features to protect grid environment from sudden failures and network attacks. SHAPE is a multi-agent model that uses the autonomic computing architecture to automate fault tolerance and security processes in the Grid. SHAPE consists of network, software, hardware and security agents to monitor the system for failures and security attacks. Agents raise alerts for any abnormal events that may occur in the system. For these agents to have self-learning capabilities, machine learning approaches like reinforcement learning and support vector machine are used. RL approach is used in automating the fault detection and correction. Fault agents work in twofold, one is to get the best resource based upon the resource failure history and second to take best possible action for a particular failure. This is decided by the reward value calculated based upon the state of failure and the best possible action taken to correct that failure. SVM is used to implement security agents to provide self-updating profile for detecting intrusions and to safeguard the system from various network attacks. Training data for SVM is formed using existing IDS (snort) rules. SVM needs a good amount of training before it starts detecting new definition of existing known attacks. Once training data is in place, the actual data from the preprocessors of IDS are converted to a format known to the LibSVM library. This data acts as the testing data. Testing data is verified with reference to the trained data and takes proper action to provide security to the grid environment. In addition to this, SHAPE provides the hardware driver hardening feature to harden all the drivers when new node is registered within the grid environment. This reduces the occurrence of failures caused by hardware. Chapter 4 mention details about the experiments and results for validating SHAPE. This chapter starts with listing down of various hardware and software require-

ments. Experiments are performed to validate self-healing and self-protection based upon certain standard metrics. For self-healing SHAPE is verified with respect to- Throughput, Turnaround time, Waiting time and Failure rate. For self-protection, metrics considered for validation are- Detection rate, False positive rate and Accuracy.

To summarize, this thesis has laid the foundation for multi-agent based autonomic self-healing and self-protection model for Grid computing called SHAPE. SHAPE has self-learning capabilities using the reinforcement learning and support vector machine algorithms. SHAPE adapts itself with the changes in Grid environment and protects the system from failures and security attacks. With these novel contributions, this thesis opens up opportunities for future research in relation to managing the uncertainty and dynamism in distributed computing systems and computational clouds.

5.2 Future work

This thesis improves the understanding of self-healing and self-protection in Grid computing environment and advances the state-of-the-art through its contributions.The investigations conducted in this thesis reveal several areas in grid fault-tolerance and security, where more work needs to be done. Moreover, the contributions of this thesis have led to new challenges that should be addressed through further research. This section briefly describes some of these challenges within the scope of the thesis.

5.2.1 Self-healing enhancements

Existing self-healing capabilities of SHAPE includes a proactive approach to reduce hardware failures using smart driver hardening, to monitor nodes and rank the nodes based upon their past failure rate and to choose the best action to correct failures. There is still space for new additions and optimizations within the existing system.

- To include additional best approaches into SHAPE. SHAPE being based upon the component-based architecture, new components can be easily added to perform an additional function. Adding checkpointing and conditional replication capabilities into SHAPE will be additional plus point. These may reduce the performance of the system to some extent but will be very useful in mission critical applications where chances for errors is negligible.

- Replication and checkpointing approaches can be optimized based upon the already gathered data using reinforcement learning. Replicas and checkpoints count can be reduced using RL approach and thus decrease load on actual grid environment.

- Improve RL approach to speed up fault detection time. Q-learning based approach is modified for distributed environment and this can be further improved in a number of ways. Another way can be to use conveniently chosen heuristic function, which would select the appropriate actions to perform in order to guide exploration during the learning process.

5.2.2 Self-protection enhancements

With SHAPE, much good work is done to eliminate possibilities of security attacks to the grid environment. Few enhancements possible as part of future work are:

- Present SHAPE detects new attacks to the extent if it is variant of already known attack. SVM help SHAPE to identify the different signature of the same attack/worm. To detect complete novel attacks, unsupervised algorithms can be taken into consideration.
- Verify the behaviour of SHAPE using other kernel function of SVM like polynomial kernel, Markov kernel, and hyperbolic tangent.
- Dynamic key management scheme can be explored with optimized communication between the agents. In this, keys are pre-distributed and dynamically updated periodically. This will help to secure agent communication.

5.2.3 Integration and validation of SHAPE with clouds

Cloud Computing [123] has emerged as the next generation platform for hosting business and scientific applications. It offers infrastructure, platform, and software as services that are made available as on-demand and subscription-based services in a pay-as-you-go model to users. The key characteristics of cloud include scalability and reliability, which ensure the consumers of cloud services that the cloud infrastructure is robust and will always be available at any time.

SHAPE can also be utilized to support self-healing and self-protecting in clouds. In future, SHAPE can be deployed across multiple platforms using Aneka

[24]. Aneka can be used to integrate different cloud and grid environments. Different experiments can then be performed to test the behaviour of SHAPE on different systems. Figure 5.1 shows the draft environment setup in that case.

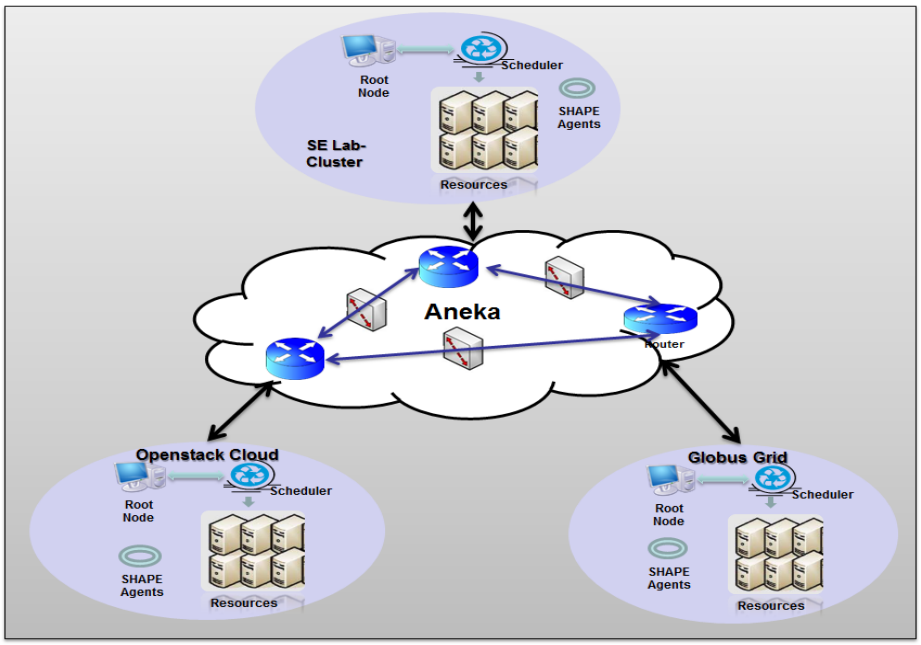


Figure 5.1: SHAPE integration with different systems

References

[1] *Supervised learning.* Wikipedia,http://en.wikipedia.org/wiki/Supervised_learning. [last accessed: May 2014]. 52

[2] *Unsupervised learning.* Wikipedia,http://en.wikipedia.org/wiki/Unsupervised_learning. [last accessed: May 2014]. 52

[3] Oracle 10g database. *ODM based intrusion detection system.* www.oracle.com/technology/products/bi/odm/pdf/odm_based_intrusion_detection-paper_1205.pdf, 2005. 47

[4] Bendahmane A, Essaaid M, Moussaoui AE, and Younes A. *Grid Computing Security Mechanisms:State-of-The-Art Support Systems.* Multimedia Computing and Systems, ICMCS, pp. 535-540, IEEE, 2009. 46

[5] Bouteiller A, Herault T, Bosilca G, and Dongarra JJ. *Correlated set coordination in fault tolerant message logging protocols.* Proceedings of the 17th International Conference on Parallel Processing-Vol. Part II, Ser. EuroPar'11, Springer-Verlag, Berlin, Heidelberg,pp. 51-64, 2011. 31

[6] Chervenak A, Foster I, Kesselman C, Salisbury C, and Tuecke S. *The Data Grid: Towards an Architecture for the Distributed Management and*

Analysis of Large Scientific Datasets. Journal of Network and Computer Applications;23(3):187-200, 2001. 3

[7] Chien A, Calder B, and Elder S. *ENTROPIA: architecture and performance for an Enterprise desktop grid system*. Journal of Parallel and Distributed Computing;65:597-610, 2003. 46

[8] Ganek A. *Overview of Autonomic Computing: Origins,Evolution, Direction*. CRC Press, 2004. 7

[9] Jain A and Shyamasundar RK. *Failure Detection and Membership Management in Grid Environments*. Fifth International Workshop on Grid Computing (GRID'04) pp. 44-52, 2004. 25

[10] Kadav A, Renzelmann MJ, and Swift MM. *Tolerating hardware device failures in software*. SOSP, pp. 11-14, Oct 2009. 97

[11] Nadalin A, Kaler C, Hallam-Baker P, and Monzillo R. *Web services security: X.509 token profile v1.0. Technical report*. IBM, Microsoft, VeriSign, Sun, 2004. 42

[12] Ng A. *Support Vector Machine*. CS229 Lecture notes, Stanford University, http://cs229.stanford.edu/notes/cs229-notes3.pdf. [last accessed: May 2014]. 53, 54

[13] Russo A and Cigno RL. *Push/pull protocols for streaming in p2p systems*. Proceedings of the 28th IEEE International Conference on Computer Communications Workshops, Brazil, 2009. 25

[14] Schulter A, Navarro F, Koch F, and Westphall C. *Towards Gridbased Intrusion Detection*. Network Operations and Management Symposium, NOMS, 10th IEEE/IFIP, pp. 14, 2006. 47

[15] Zolnowski AP. *JADE-PKI 1.0 Manual*. JADE Documentation http://jade.tilab.com/doc/tutorials/PKI_Guide.pdf, September 2012. 75, 76

[16] Atkinson B, Della-Libera G, Hada S, Hondo M, Hallam-Baker P, and Klein J et al. *Web services security (ws-security)*. Technical report, IBM, Microsoft, VeriSign, 2002. 42

[17] Claudel B, Palma ND, Lachaize R, and Hagimont D. *Self-protection for Distributed Component-Based Applications*. Springer-Verlag Berlin Heidelberg, 2006. 37

[18] Khoo B and Veeravalli B. *Pro-active failure handling mechanisms for scheduling in grid computing environments*. J. Parallel and Distributed Computing;70(3):189-200, 2010. 34

[19] Schroeder B and Gibson GA. *A Large-Scale Study of Failures in High Performance Computing Systems*. Dependable and Secure Computing, IEEE Transactions;7(4):337-351, 2010. 21

[20] Bergenti F Bellifemine F, Caire G, and Poggi A. *Jade A Java Agent Development Framework*. Multiagent Systems, Artificial Societies, and Simulated Organizations Volume 15, pp. 125-147, 2005. 74

[21] Ebrahimi BP, Bertels k, Vassiliadis S, and Sigdel K. *Matchmaking within multi-agent systems*. proceeding of ProRisc, November 2004. 13

[22] Briquet C and Marneffe PA. *Grid Resource Negotiation: Survey with machine learning perspective.* Proceedings of the 24th IASTED International Conference on Parallel and Distributed Computing and Networks; pp.14-16, Feb 2006. 50

[23] Kesselman C, Foster I, and Karonis NT. *Managing security in high performance distributed computations.* Technical report, The Globus Alliance http://www.globus.org, 1997. 43, 44

[24] Vecchiola C, Chu X, Mattess M, and Buyya R. *ANEKA-Integration of private and public clouds.* Cloud Computing: Principles and Paradigms, John Wiley & Sons, Inc, 2011. 126

[25] Wang C, Vazhkudai SS, Ma X, and Mueller F. *Comparative Analysis of Fault Tolerance Techniques in Grid Environment.* Fault Tolerance for Job Input Data in HPC Environments, Chapter in Handbook on Data Centers, 2014. 21

[26] Chang CC and Lin CJ. *LIBSVM – A Library for Support Vector Machines.* LIBSVM team: http://www.csie.ntu.edu.tw/~cjlin/libsvm/, May 2014. 103

[27] Hsieh CC. *Optimal task allocation and hardware redundancy policies in distributed computing systems.* Eur. J. Operational Res.;147(2):430-47, 2003. 33

[28] Anirban Chakrabarti. *Grid Computing Security.* Springer, Chapter-6;pp105-131, 2007. 47

[29] Ghosh D., Sharman R., Rao H.R., and Upadhyaya S. *Self-Healing Systems-Survey and Synthesis*. Decision Support Systems;42:2164-2185, 2007. 20

[30] Meyer D. *Support Vector Machines*. http://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf, March 2014. 103

[31] Bartolini DB. *Reinforcement Learning in Autonomic Computing*. Project assignment for the Advanced Topics of Machine Learning, 2012. http://home.deib.polimi.it/bartolini/pub/informal-tr/bartolini-2012-atml-rlac.pdf Last accessed: May 2014. 58

[32] Diaz-Verdejo, Jesus E, Garcia-Teodoro, Pedro, Munoz P, and Macia-Fernandez G et al. *A Snort-based approach for the development and deployment of hybrid IDS*. IEEE Latin America Transactions;5(6):386-92, Oct 2007. 49

[33] Alpaydin E. *Introduction to Machine Learning*. The MIT Press, second edition; pp. 1-46, 2010. 50, 51

[34] Feller E, Mehnert-Spahn J, Schoettner M, and Morin C. *Independent checkpointing in a heterogeneous grid environment*. Future Generation Computer Systems;28(1):163-170, 2012. 31

[35] Meneses E, Mendes CL, and Kale LV. *Team-based message logging: preliminary results*. Cluster, Cloud and Grid Computing, CCGrid, 10th IEEE/ACM International Conference, pp. 697-702, May 2010. 31

[36] Raju E and Sravanthi K. *Network intrusion detection using Support Vector*

Machines. International Journal of Computer Science and Management Research;2(1), Jan 2013. 53

[37] Elnozahy ENM, Alvisi L, Wang YM, and Johnson DB. *A survey of rollback recovery protocols in message-passing systems*. ACM Computing Surveys;34(3):375-408, 2002. 29

[38] Kon F, Roman, and Liu P. *Monitoring, security and dynamic configuration with the dynamic TAO reflective ORB*. IFIP/ACM International Conference on Distributed Systems Platforms,pp. 121-143, New York, United States, 2000. 46

[39] Leu F, Li M, and Lin J. *Intrusion Detection based on Grid*. ICCGI, 2006. 47

[40] Fermilab. *Strong authentication at fermilab*. Technical report, Fermilab, 2004. 42

[41] Khan FJ, Qureshi K, and Nazir B. *Performance evaluation of fault tolerance techniques in grid computing system*. Computers and Electrical Engineering;36:1110-22, Nov 2010. 20

[42] Cheswick FR and Bellovin SM. *Firewalls and internet security: Repelling the wily hacker*. Technical report, Addison-Wesley, Reading, MA, 1994. 46

[43] Caire G. *JADE programming for beginners*. JADE documentation http://www.cs.uu.nl/docs/vakken/map/JADEProgramming-Tutorial-for-beginners.pdf, June 2009. 74

[44] Tesauro G. *Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies.* Internet Computing, IEEE, 11(1):2230, feb 2007. 58

[45] Wen GAO, Mingyu CHEN, and Takashi NANYA. *A Faster Checkpointing and Recovery Algorithm with a Hierarchical Storage Approach.* Eighth International Conference HPCASIA, 2005. 30

[46] Suryawanshi GR and Vanjale SB. *Mobile Agent for Distributed Intrusion detection System in Distributed System.* Procedings in International Journal of Artifical Intelligence and Computational Research (IJAICR), June 2010. 50

[47] Hoffmann H and Eastep J. *Application heartbeats a generic interface for specifying program performance and goals in autonomous computing environments.* Proceeding of the 7th international conference on Autonomic computing, ICAC, pp. 79-88, New York, NY, USA, ACM, 2010. 26

[48] Hu H, Guo S, and Yang B. *Cost-oriented task allocation and hardware redundancy policies in heterogeneous distributed computing systems considering software reliability.* Comput. Ind. Eng.;56(4):1687-96, 2009. 33

[49] Team HTCondor. *HTCondor Version 8.2.1 Manual.* Center for High Throughput Computing, University of WisconsinMadison, July 2014. 29

[50] Brahmi I, Yahia SB, and Poncelet P. *A Snort-Based Mobile Agent For A Distributed Intrusion Detection System.* SECRYPT 2011 - Proceedings of the International Conference on Security and Cryptography, Seville, Spain, pp. 18 - 21, July 2011. 49

[51] Chopra I and Singh M. *Agent based Self-Healing System for Grid Computing.* ICWET'10 Proceedings of the International Conference and Workshop on Emerging Trends in Technology; pp. 31-35, ACM, New York, 2010. 67

[52] Chopra I and Singh M. *Analysing the need for Autonomic Behaviour in Grid Computing.* Computer and Automation Engineering (ICCAE), Vol 1, pp. 535-539, IEEE, 2010. 66

[53] Chopra I and Singh M. *Automating the fault tolerance process in Grid Environment.* International Journal of Computer Science and Information Security;8(7):224-30, Oct 2010. 67

[54] Chopra I and Singh M. *Implementing Self-Protection in distributed grid environment.* Scientific International Journal for Parallel and Distributed Computing;11(4):373-383, Dec 2010. 40, 68

[55] Chopra I and Singh M. *SHAPE - A Model for Self-healing and Self-Protection in Complex Distributed Systems.* The Journal of Supercomputing;67:585-613, Feb 2014. 70

[56] Foste I, Karonis NT, Kesselman C, Koenig G, and Tuecke S. *A secure communications infrastructure for high-performance distributed computing.* Technical report, The Globus Alliance http://www.globus.org, 1997. 44, 45

[57] Foster I and Kesselman C. *The Grid 2: Blueprint for a New Computing Infrastructure.* Second Edition, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, pp. 15-51, 1999. 5, 21, 40, 42, 43, 45, 46

[58] Foster I, Kesselman C, Tsudik G, and Tuecke S. *A security architecture for computational grids.* Technical report, The Globus Alliance http://www.globus.org, 1998. 40

[59] Foster I, Kesselman C, Nick J, and Tuecke S. *Grid Services for Distributed System Integration.* Journal Computer;35(6):37-46, 2002. 2

[60] Foster I, Kesselman C, Nick JM, and Tuecke S. *The physiology of the Grid.* Global Grid Forum, June 2002. 2

[61] Foster I, Kesselman C, and Tuecke S. *The Anatomy of the Grid Enabling Scalable Virtual Organizations.* www.globus.org, 2004. 2

[62] Foster I, Kesselman C, Tuecke S, Pearlman L, and Welch V. *A community authorization service for group collaboration.* Technical report, The Globus Alliance http://www.globus.org, 2002. 43

[63] Foster I, Yong Zhao, Raicu I, and Shiyong Lu. *Cloud Computing and Grid Computing 360-Degree Compared.* Grid Computing Environments Workshop,GCE08 ,pp.1,10, 12-16 Nov, 2008. 5

[64] IBM. *An architectural blueprint for autonomic computing.* IBM Press, June 2006. http://www-03.ibm.com/autonomic/pdfs/AC8, 9

[65] IBM. *Stopping insider attacks how organizations can protect their sensitive information.* http://ibm.com/services, September 2006. 40

[66] Egwutuoha IP, Levy D, Selic B, and Chen S. *A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems.* Journal of Supercomputer;65:1302-26, 2013. 21

[67] Egwutuohaa IP, Chenb S, Levya D, Selica B, and Calvoa R. *Cost-oriented proactive fault tolerance approach to high performance computing (HPC) in the cloud.* International Journal of Parallel, Emergent and Distributed Systems;29(4):363-378, Jan 2014. 29

[68] Dowling J, Cunningham R, Curran E, and Cahill V. *Collaborative Reinforcement Learning of Autonomic Behaviour.* Database and Expert Systems Applications,pp. 700 - 704, Sept 2004. 61

[69] Frey J, Tannenbaum T, Livny M, Foster I, and Tuecke S. *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* 10th IEEE International Symposium on High Performance Distributed Computing, 2001. 29

[70] Gomez J, Gil C, Padilla N, Banos R, and Jimenez C. *Design of a Snort-Based Hybrid Intrusion detection System.* IWANN Part II, LNCS 5518, pp. 515-522, 2009. 48

[71] Park J, Jung J, Piao S, and Lee E. *Self-healing Mechanism for Reliable Computing.* International Journal of Multimedia and Ubiquitous Engineering;3(1):75-86, Jan 2008. 11

[72] Seo J, Lee C, Shon T, and Moon J. *SVM Approach with CTNT to Detect DDoS Attacks in Grid Computing.* Grid and Cooperative Computing - GCC, Lecture Notes in Computer Science;Vol(3795),pp. 59-70, Dec 2005. 53

[73] Wang J, Liu X, and Chien A. *Empirical Study of Tolerating Denial-of-*

Service Attacks with a Proxy Network. Proceedings of the 14th conference on USENIX Security Symposium, 2005. 38

[74] Weston J. *Support Vector Machine (and Statistical Learning Theory)*. NEC Labs America, Princeton, USA, http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf. [last accessed: May 2014]. 55

[75] Exposito JA, Ametller J, and Robles S. *Configuring the JADE HTTP MTP*. JADE Documentation http://jade.tilab.com/doc/tutorials/HTTP_UAB.html, 2010. 74

[76] Durand JC. *Grid Computing*: *A Conceptual and Practical Study*. University of Lausanne, November 2004. 2, 4

[77] Platt JC. *Advances in kernel methods*. Cambridge, MA, USA: MIT Press; pp. 185-208, 1999. 56

[78] Abawajy JH. *Fault-Tolerant Scheduling Policy for Grid Computing Systems*. IPDPS, 2004. 33, 34

[79] Crichlow JM. *An Introduction to Distributed and Parallel Computing*. Computing in Science and Engineering; pp. 61-71, 2003. 2

[80] Levine John, LaBella Richard, Owen Henry, Contis Didier, and Culver Brian. *The use of honeynets to detect exploited systems across large enterprise networks*. IEEE Systems Man and Cybernetics SocietyInformation Assurance Workshop pp:9299, 2003. 50

[81] Decker K, Sycara K, and Williamson M. *Middle-agents for the Internet.* Proc IJCAI, pp. 578-584, 1997. http://www.cs.cmu.edu/~softagents/papers/middleagents.pdf. 13

[82] Djemame K, Gourlay I, Padgett J, Birkenheuer G, Hovestadt M, and Kao O et al. *Introducing risk management into the grid.* Proceedings of the 2nd IEEEInternational Conference on e-Science and Grid Computing, IEEEComputer Society, Amsterdam, Netherlands, Dec 2006. 35

[83] Djemame K, Padgett J, Gourlay I, and Armstrong D. *Brokering of risk-aware service level agreements in grids.* Concurrency and Computation: Practice and Experience;23(7):1558-82, Sept 2011. 35

[84] Hwang K, Cai M, Chen Y, and Qin M. *Hybrid Intrusion Detection with Weighted Signature Generation Over Anomalous Internet Episodes.* IEEE Transactions on Dependable and Secure Computing;4(1):41-55, Jan 2007. 49

[85] Krauter K, Buyya R, and Maheswaran M. *A taxonomy and survey of grid resource management systems for distributed computing.* John Wiley and Sons Ltd, 2001. 3

[86] Srinivasa K, Siddesh G, and Cherian S. *Fault-tolerant middleware for grid computing.* Proc. of 12th IEEE International Conference on High Performance Computing and Communications, Melbourne, Australia;pp. 635-640, Sept 2010. 32

[87] Frederick KK, Ronald W, Northcutt RS, Zeltser L, and Winters S. *INSIDE - Network Perimeter Security.* New Riders, 2003. 45

[88] Murphy KP. *Machine Learning A Probabilistic Perspective*. The MIT Press Cambridge, Massachusetts London, England, May 2012. 51, 52

[89] Slabeva KS, Wozniak T, and Ristol S. *Grid and Cloud Computing-A Business Perspective on Technology and Applications*. Hardcover Springer, 2010. 5

[90] Kaelbling LB and Moore AW. *Reinforcement Learning: A Survey*. Journal of Artificial Intelligence Research; Vol4:237-85, 1996. 52

[91] Wuu LC, Hung CH, and Chen SF. *Building intrusion pattern miner for Snort network intrusion detection system*. Journal of Systems and Software;80(10):1699-715, 2007. 49

[92] Adamski M, Chmielewski M, Fonrobert S, Gowdiak A, Lewandowski B, and Nabrzyski J et al. *D6.1 requirements document*. Technical report, GridLab - http://www.gridlab.org, 2001. 43, 44

[93] Ali Aydin M, Zaim AH, and Ceylan KG. *A hybrid intrusion detection system design for computer network security*. Computers and Electrical Engineering;35:517-26, April 2009. 48

[94] Amoon M. *A development of fault-tolerant and scheduling system for grid computing*. GESJ Computer Science and Telecommunications, 2011. 49

[95] Amoon M. *Fault tolerance in grids using job replication*. International Journal of Computing;11(2):115-21, 2012. 32

[96] Amoui M, Salehie M, Mirarab S, and Tahvildari L. *Adaptive Action Selection in Autonomic Software Using Reinforcement Learning*. Proc. of the

Fourth International Conference on Autonomic and Autonomous Systems, pp.175-181, 2008. 60

[97] Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, and Konwinski A et al. *Above the clouds: A berkeley view of cloud computing.* Technical Report EECS-2009-28, EECS at UC Berkeley, 2009. 5

[98] Benattou M. and Tamine K. *Intelligent Agents for Distributed Intrusion Detection System.* World Academy of Science, Engineering and Technology International Journal of Computer, Information, Systems and Control Engineering Vol:1 No:6, 2007. http://waset.org/publications/14763/intelligent-agents-for-distributed-intrusion-detection-system. 47

[99] Chetty M and Buyya R. *Weaving computational grids: how analogous are they with electrical grids*?. Computing in Science and Engineering; pp. 61-71, 2002. 2

[100] Chtepen M, Dhoedt B, Turck FD, and Demeester P. *Evaluation of replication and rescheduling heuristics for grid systems with varying resource availability.* Parallel and Distributed Computing and Systems, 2006. 34

[101] Humphrey M and Thompson M. *Security Implications of Typical Grid Computing Usage Scenarios.* Cluster Computing;5(3):95-103, 2002. 40

[102] Maggio M, Hoffmann H, and Leva A. *A Comparison of Autonomic Decision Making Techniques.* Computer Science and Artificial Intelligence Laboratory Technical Report, MIT, 2011. 27

[103] Maggio M, Hoffmann H, and Leva. *Controlling software applications via resource allocation within the heartbeats framework.* In Proceeding of the 49th international conference on decision and control, Atlanta, USA, IEEE, 2010. 26

[104] Parashar M and Hariri S. *Autonomic Computing: An Overview.* Springer-Verlag Berlin Heidelberg, 2005. 7

[105] Rahman M, Ranjan R, Buyya R, and Benatallah B. *A taxonomy and survey on autonomic management of applications in grid computing environments.* Concurrency and Computation: practice and experience, John Wiley & Sons, Ltd, 2011. 2, 3

[106] Witold M and lkowicz J. *A Grid Aware Intrusion Detection System.* Technical University of Denmark, IMM-THESIS, 2007. 47

[107] Tolba MF, Abdel-Wahab MS, Taha LA, and Al-Shishtawy AM. *GIDA: Toward Enabling Grid Intrusion Detection System.* Proc. Conference on Cluster Computing and Grid (CCGrid), Cardi (Wales), 2005. 47

[108] Huhns MN., Holderfield V.T., and Gutierrez R.L. *Robust software via agent-based redundancy.* AAMAS'03, 2003. 36

[109] Iyatiti Mokube and Michele Adams. *Honeypots: concepts, approaches, and challenges.* 45th annual southeast regional conference, ACM-SE 45, pages 321326 ACM, 2007. 50

[110] Navimipour NJ, Rahmani AM, Navin AH, and Hosseinzadeh M. *Resource*

discovery mechanisms in grid systems:A survey. Journal of Network and Computer Applications;41:389-410, May 2014. 3

[111] Hinnelund P. Autonomic computing - a method for automated systems management systems management using distributed local control. Master's thesis, Royal Institute of Technology, March 2004. 6

[112] Horn P. Autonomic computing:ibm's perspective on the state of information technology, October 2001. http://www.research.ibm.com/autonomic/. 9, 20

[113] Mell P and Grance T. *The NIST Definition of Cloud computing*. National Institute of Standards and Technology. Available online at:http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-defv15.doc, 2009. 5

[114] Stelling P, Foster I, Kesselman C, Lee C, and Laszewski G. *A Fault Detection Service for Wide Area Distributed Computations*. High Performance Distributed Computing, 1998. 27

[115] Townend P and Xu J. *Fault Tolerance within a Grid Environment*. In Proceedings of the UK e-Science All Hands Meeting AHM;272-275, 2003. 20

[116] Liang PENG and Lip Kian NG. *N1GE6 Checkpointing and Berkeley Lab Checkpoint/Restart*. SUN microsystem, Asia pacific science and technology center, 2004. 29

[117] Honeynet Project. *Know Your Enemy:Honeynets*. What a honeynet is,

its value, overview of how it works, and risk/issues involved.,http://old.honeynet.org/papers/honeynet/. [last accessed: May 2014]. 50

[118] The Snort Project. *SNORT Users Manual*. Cisco and/or its afliates, April 2014. 47

[119] Jiang Q, Luo Y, and Manivannan D. *An optimistic checkpointing and message logging approach for consistent global checkpoint collection in distributed systems*. Journal of Parallel and Distributed Computing;68(12):1575-89, 2008. 31

[120] Alsoghayer R and Djemame K. *Resource failures risk assessment modelling in distributed environments*. The Journal of Systems and Software;88:42-53, 2014. 21

[121] Alsoghayer R and Djemame K. *Resource failures risk assessment modelling in distributed environments*. The Journal of Systems and Software;Vol88:pp.42- 53, Feb 2014. 21, 36

[122] Berwick R. *An Idiot's guide to Support vector machines (SVMs)*. Lecture Slides,http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf, 2003. [last accessed: May 2014]. 53

[123] Buyya R, Yeo C, Venugopal S, Broberg J, and Brandic I. *Cloud computing and emerging IT plateforms Vision, hype and reality for delivering computing as 5th utility*. Future generation computer systems, 2009. 6, 125

[124] Buyya R, Yeo CS, Venugopal S, Broberg J, and Brandic I. *Cloud computing and emerging it platforms: Vision, hype, and reality for delivering comput-*

ing. 5th utility, Future Generation Computer Systems; 25(6):599-616, 2009. 5

[125] Buyya R, D.A., and Giddy J. *Grid Resource Management, Scheduling and Computational Economy.* 2nd International Workshop on Global and Cluster Computing, 2000. 28

[126] Buyya R, D.A., and Giddy J. *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid.* 4th International Conference on High Performance Computing, 2000. 28

[127] Garg R and Singh AK. *Fault Tolerance in Grid Computing: State of Art and Open Issues.* International Journal of Computer Science and Engineering Survey;2(1):88-97, 2011. 25

[128] Riesen R, Ferreira K, and Stearley J. *See Applications Run and Throughput Jump: The Case for Redundant Computing in HPC.* Proceedings of the 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 29-34, 2010. 33

[129] Bawa RK and Singh R. *Comparative Analysis of Fault Tolerance Techniques in Grid Environment.* International Journal of Computer Applications;41(1);21-5, 2012. 21

[130] Renesse RV, Minsky Y, and Hayden M. *A gossip-style failure detection service.* Technical Report TR98-1687, Cornell University, Computer Science, 1998. 27

[131] Hariri S and Qu G. *Quality-of-Protection (QoP)-An Online Monitoring and Self-Protection Mechanism.* IEEE Journal on selected areas in communications;23(10):1983-93, 2005. 37

[132] Kenny S and Coghlan B. *Towards a Grid-Wide Intrusion Detection System.* Advances in Grid Computing, Springer, pp. 275-284, 2005. 47

[133] Krishnan S. *An Architecture for Checkpointing and Migration of Distributed Components on the Grid.* Department of Computer Science, Indiana University, 2004. 29, 30

[134] Maciej S, Adamus S, Bugaa S, and Szmit A. *Anomaly detection 3.0 for Snort.* Snort AD Project http://anomalydetection.info/, 2012. 83

[135] Mukkamala S, Janoski G, and Sung A. *Intrusion detection using neural networks and support vector machines.* Proceedings of the International Joint Conference on Neural Networks IJCNN;Vol(2), pp. 1702-1707, May 2002. 53

[136] Russell S and Norvig P. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. 58

[137] Singh S, Sarkar M, Roy S, and Mukherjee N. *A Survey on Application of Machine Learning to Resource Management in Grid Environment.* Lecture Notes on Software Engineering, Vol. 1, No. 2, May 2013. 50

[138] Mulay SA, Devale PR, and Garje GV. *Intrusion Detection System using Support Vector Machine and Decision Tree.* International Journal of Computer Applications;3(3):40-3, June 2010. 53

[139] Kotsiantis SB. *Supervised Machine Learning: A Review of Classifications Techniques*. Proceedings of the conference on Emerging Artificial Intelligence Applications in Computer Engineering,pp. 3-24, Oct 2007. 52, 53

[140] Hashemi SM and Bardsiri AK. *Cloud Computing Vs. Grid Computing*. ARPN Journal of Systems and Software; 2(5):188-94, 2012. 5

[141] Snort. *Leveraging Centralized Encryption Snort-part-1*. https://edge.arubanetworks.com/article/leveraging-centralized-encryption-snort-part-1, 2000. 47

[142] Sathya SS and Babu KS. *Survey of fault tolerant techniques for grid*. Computer Science Review; 4:101-20, Feb 2010. 20

[143] Cheng S.W., Garlan D., Schmerl B., Steenkiste P., and Hu N. *Software architecture-based adaptation for grid computing*. The 11th IEEE Conference on High Performance Distributed Computing (HPDC'OZ). Edinburgh, Scotland, 2002. 36

[144] Altameem T. *Fault Tolerance Techniques in Grid Computing Systems*. International Journal of Computer Science and Information Technologies;4(6):858-62, 2013. 32, 33

[145] Ryutov T, Neumann C, and Zhou L. *Integrated Access Control and Intrusion De-tection (IACID) Framework for Secure Grid Computing*. Technical Report, University of Southern California, 2005. 47

[146] TanvirM, Heinz H, Ian WS, and Peake D. *An Agent Oriented Proactive*

Fault-tolerant Framework for Grid Computing. e-Science and Grid Computing, pp. 3-11, 2005. 28

[147] Ding U, Xiao M, and Liu A. *Research And Implementation On Snort-Based Hybrid Intrusion Detection System.* Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, Baoding, pp12-15, July 2009. 48

[148] Katkar V and Bhirud SG. *Novel DoS/DDoS Attack Detection and Signature Generation.* International Journal of Computer Applications;47(10):18-24, June 2012. 49

[149] Sekhri V. *CompuP2P: A light-weight architecture for Internet computing.* MS Thesis, Iowa State University, Iowa, 2005. 20

[150] Welch V, Foster I, Kesselman C, Mulmo O, Pearlman L, and Tuecke S et al. *X.509 proxy certificates for dynamic delegation.* 3rd Annual PKI R&D Workshop, 2004. 42

[151] Welch V, Novotny J, and Tuecke S. *An online credential repository for the grid: Myproxy.* Technical report, The Globus Alliance http://www.globus.org, 2001. 45

[152] Ahmed W and Wu YW. *A survey on reliability in distributed systems.* Journal of Computer and System Sciences;79(8):1243-55, Dec 2013. 11

[153] Noble WS. *What is a support vector machine?* Nature Biotechnology;Vol.24(2):pp.1565-7, Dec 2006. 53

[154] Luo Y and Manivannan D. *HOPE: A Hybrid Optimistic checkpointing and selective pessimistic message logging protocol for large scale distributed systems.* Future Generation Computer Systems;28:1217-35, 2012. 31

[155] Wang Y, Wong J, and Miner A. *Anomaly intrusion detection using one class SVM.* Proceeding of: Information Assurance Workshop;pp. 358-364, June 2004. 53

[156] Geng Yang, Chunming Rong, and Yunping Dai. *A Distributed Honeypot System for Grid Security.* Grid and Cooperative Computing Lecture Notes in Computer Science Volume 3032, pp 1083-1086, 2004. 50

[157] Jin YC and Sendhoff B. *Pareto-based multiobjective machine learning: An overview and case studies.* IEEE Transactions on Systems, Man and Cybernetics, part C: Applications and Reviews,38(3):397-415, May 2008. 50

[158] Zhung YF, Sun J, and Ma JB. *Self Management Model Based on multiagent and Worm Techniques.* CCECE, 2004. 22

[159] Ghahramani Z. *Unsupervised Learning.* Advanced Lectures on Machine Learning LNAI, Sept 2004. 51

Publications

1. Chopra I and Singh M. SHAPE - A Model for Self-healing and Self-protection in Complex Distributed Systems. The Journal of Supercomputing;Vol. 67(2):pp585-613,Springer, Feb 2014.*[Indexing: SCI, SciSearch, ACM digital library]*

2. Chopra I and Singh M. Implementing Self-Protection in distributed grid environment. Scientific International Journal for Parallel and Distributed Computing;Vol. 11(4):373-383, Dec 2010.*[Indexing: DOAJ, EBSCO]*

3. Chopra I and Singh M. Automating the fault tolerance process in Grid Environment. International Journal of Computer Science and Information Security;Vol. 8(7):224-30, Oct 2010. *[Indexing: Google Scholar, Scientific Commons, SCIRUS, CiteSeerX]*

4. Chopra I and Singh M. Analysing the need for Autonomic Behaviour in Grid Computing. Computer and Automation Engineering (ICCAE), Vol 1,pp535-539, IEEE, 2010. *[Indexing: IEEEXplore]*

5. Chopra I and Singh M. Agent based Self-Healing System for Grid Computing. ICWET'10 Proceedings of the International Conference and Workshop on Emerging Trends in Technology; pp31-35, ACM, New York, 2010. *[Indexing: ACM digital Library]*