Efficient Parallel Architecture of SWAF for Multi-core Sysrems

By
Fauzia Noureen
2008-NUST-MS-PhD-IT-34

Supervisor
Dr. Hafiz Farooq Ahmad
NUST-SEECS

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Technology (MS IT)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(April 2011)

Approval

It is certified that the contents and form of the thesis entitled "**Efficient Parallel Architecture of SWAF for Multi-core Sysrems**" submitted by **Fauzia Noureen** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Hafiz Farooq Ahmad**

Signature: ________________
Date: ________________

Committee Member 1: **Sir Ibrar Ahmed**

Signature: ________________
Date: ________________

Committee Member 2: **Sir Aatif Kamal**

Signature: ________________
Date: ________________

Committee Member 3: **Sir. Ammar Karim**

Signature: ________________
Date: ________________

Abstract

Multi-core systems and multi-core servers are present in almost all data centers now-a-days. All major processor vendors are producing multi-core chips. These multi-core systems are commonly used to deploy high end applications such as firewalls. However the capacity of a system cannot by fully utilized unless and until application running on the system is not capable of utilizing it. So multi-threaded application architecture is need for mission critical systems like SWAF (Semantic Web Application Firewall). Fully parallelized application running on multi-core system achieves high performance. To design fully parallelized SWAF, all different modules need to run concurrently. Furthermore each module has to have its own thread pool containing multiple numbers of threads which are responsible for performing tasks specific to that module. The research challenges faced while designing such architecture includes reducing the SWAF's complexity, minimizing the increased inter-modular communication and most importantly usage of different optimized techniques to remove sequential dependencies among different modules. Each module places intermediate result in a memory buffer so that the other module can read it and perform further tasks on it. JVM (Java Virtual Machine) manages and distributes the thread pools to the available cores of the system and assures to run each module on different core of the machine concurrently. The system has been evaluated on both low and high loads to test the performance. At low load of requests, old architecture gave better results than the proposed one. The reason was the increased communication between different modules of SWAF. The test results at high load of requests proved that multi-core SWAF gave consistent behavior where as old SWAF code become irresponsive at high loads.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at National University of Sciences & Technology (NUST) School of Electrical Engineering & Computer Science (SEECS) or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Fauzia Noureen**

Signature: ________________

Acknowledgment

First and foremost, I am immensely thankful to Almighty Allah for letting me pursue and fulfill my dreams. Nothing could have been possible without His blessings.
After that I am thankful to my family and friends for supporting me in every ways possible to complete my work. I am grateful to my thesis supervisor, Dr. Hafiz Farooq Ahmed, for his continuous support and guidance during this journey. He gave me motivation whenever I felt difficulties in my work. Without his guidance and encouragement I would not be able to complete my thesis. He was always available for me during the difficulties I faced regarding any matter. He was really very generous and patient during this thesis time span.
This thesis would not have been possible without the expert opinions of my Co-supervisor, Sir Ibrar Ahmed, who was a great source of inspiration and guidance. His work experience helped me a lot during my thesis work. His experience and vision helped me resolving my problems during work. I am really grateful to him because he spared time from his hectic industry routines. I am also thankful to my committee members Sir Aatif Kamal and Sir Ammar Karim for their support and suggestions.
I am also thankful to SWAF Team leads Ali Hur and Abdul Razzaq for their guidance. They helped me understanding SWAF which supported me to integrate my ideas in SWAF. I am also thankful to all SWAF team members for their help and support.

Fauzia Noureen

Contents

List of Abbreviations

Abbreviations	**Descriptions**
SWAF	Semantic based Web Application Firewall
WAF	Web Application Firewall
OSI	Open Systems Interconnection
CPU	Central Processor Unit
QoS	Quality of Service
I/O	Input Output
HTTP	HyperText Transfer Protocol
URI	Uniform Resource Identifier
RFC	Request For Comment
JVM	Java Virtual Machine
FIFO	First In First Out
SQL	Structured Query Language
DT	Directory Traversal
XSS	Cross Site Scripting
DoS	Denial of Service
DDoS	Distributed Denial of Service
RMI	Remote Method Invocation
SOA	Service Oriented Architecture
VM	Virtual Machine
RAM	Random Access Memory

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

1.1 Introduction

1.1.1 Why Web Application Firewall (WAF)?

The security of web applications has become increasingly important and a secure web environment has become a high priority for e-businesses communities [1]. Traditional firewalls do not understand attacks directed at the code of the application. If the firewall sees TCP port 80, the traffic is allowed through; no matter what malicious code it may contain [2]. Web application firewalls are used to protect the application layer attacks. WAFs are capable of preventing attacks that network firewalls and intrusion detection systems cannot. In most cases WAFs do not require modification of application source code [13]. According to the statistics given by various authentic sources, such as MITRE, OWASP [3], WHITE HAT, ACUNETIX, about 75 % of information security attacks are being launched at Application layer of OSI (Open Systems Interconnection) model. A security assessment by the Application Defense Center, which included more than 250 Web applications from e-commerce, online banking, enterprise collaboration, and supply chain management sites, concluded that at least 92% of Web applications are vulnerable to some form of attack [4]. Another survey found that about 75% of all attacks against web servers target the web applications also [5]. WAFs are often called 'Deep Packet Inspection Firewall' because they look at each request and response. Unfortunately packet inspections can impose significant delays on traffic due to the complexity and size of policies [6].

Traditional firewall implementations consist of a single dedicated machine, similar to a router that sequentially applies the policy to each arriving packet. However, packet filtering can represent a significantly higher processing load

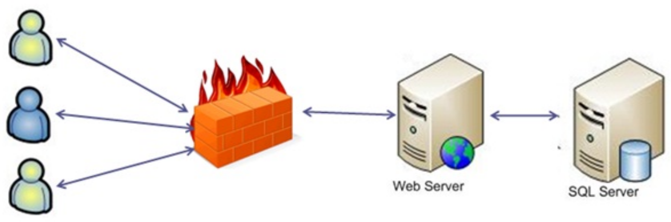


Figure 1.1: Deployment of a WAF

than routing [7]-[9]. For example, a firewall that interconnects two 100Mbps networks would have to process over 300,000 packets per second [10]. Successfully handling this traffic load becomes more difficult as policies become more complex [7], [11], [12]. An Application firewall's role is to improve application security by integrating knowledge about an application's specific security needs into elements of the IT security infrastructure [13].

1.1.2 Multi-threading on Single core vs. Multi-core

Application performance can be improved by having multiple threads in the program. Many web applications are using concept of multi-threading to improve their time to respond to a user request over single core platforms. The concept of multi-threading over single core platforms provides one significant advantage. If one thread of application is working on a time consuming task like heavy database query or disk access than instead of waiting the other thread will utilize the CPU for its working. These improved instructions application can has by using multiple threads over single core is known as 'interleave instructions stream'. In this scenario threads are not running in parallel to each other so the performance gain of multi-threading over single core is limited.

However this performance restriction is not there when multi-threaded application runs over multi-core platform. On a multi-core system, threads don't need to wait for each other. Rather they run independently over different cores of CPU[16]. The theme for such systems is that applications should be optimized in such a way that different threads run over different cores of the system.

1.2 Motivation

Research proved that the questions which are normally asked by the client before deployment of firewall includes, "Is there a significant performance loss while deploying a firewall in the network?" and "What level of security we (client) should expect without sacrificing the network performance?" [14]. It gives a motivation that in the client's perspective performance is as important as security of the application. Client is concerned that the increased security should not compromise the performance of the application.

Moore's law states, the number of transistors inside a single chip has continued to increase exponentially after every 18 months which also gives motivation that performance of applications can be improved.

Another motivation is an interesting note that the intuitive belief about security to performance i.e. the more security would result in less performance, does not always hold in the firewall testing. [6]

1.3 Research Objectives

The Web Application Firewall (WAF) functions by being positioned between the user-side client and the application server, thereby intercepting all data passed between the application server and the user. This traffic is examined by the WAF against various rules in an attempt to determine which data is valid and which is considered invalid [13] and as discussed earlier that WAFs are also known as 'deep packet inspectors' because they will in detail check the packet is malicious or not. As discussed earlier, it is proved that WAFs are the best solution for securing the web server but the performance enhancement is WAF is very important factor.

Performance is very important factor for a firewall. If a client deploys firewall in-front of an application server they needs to be sure that it won't affect the performance of the application very much. The demand for computing power continues to increase in virtually every domain, from the basic desktop systems to the high-end computing platforms [19]. The emerging multi-core architectures provide a solution to increase the performance capability on a single chip without requiring a complex system and increasing the power requirements [19]-[23]. Therefore, multi-core systems have become the dominant architecture for both desktop and high-performance platforms.

Protecting the application servers to the highest level possible without affecting the performance is desired by the industry today. This is the era of multi-core systems and multi-core servers are present in almost all data centers now-a-days. However the capacity of a system cannot by fully utilized unless and until application running on that system is not capable of utilizing it. So the Multi-core & multi-threaded architecture is need of mission critical systems like WAFs.

1.4 Thesis Organization

Chapter 2 includes the research backgrounds of this topic starting from basics of this domain moving onto the test run performed and its results and ending on the relevant work proposed by different people in this domain. Chapter 3 includes the details about the proposed methodology and how it is implemented in the system. The proposed architecture is also discussed in detail in this chapter. Chapter 4 is all about the implementation details of my thesis. Different data structures used on java code and what advantages I am gaining from them. This chapter also contains code snippets of my thesis. Chapter 5 contains evaluation and testing results of the proposed system. Different tests are included in this chapter to compare that the performance of proposed architecture is better than the previous system. Chapter 6 concludes the thesis and contains few suggestions based on what performance enhancements can be evaluated in future.

Chapter 2

Existing Techniques and Architectures

2.1 Overview

In this chapter first I discussed different reasons due to which performance of a WAF (Web Application Firewall) is so much important. After that two Laws Moore's law and Amdahl's Law is discussed. These laws shows that capabilities of processors is increasing with time, the need is to have such applications which can fully utilize the capacity of hardware processors. After that; results of a small test is shared. Test results proved that multi-threaded programs are required for multi-core platforms to get best performance results. After that different relevant architectures from different research papers are discussed with their pros and cons. I have concluded the chapter after these discussions.

2.2 Performance impact of a WAF

Performance impact may cause major concerns while deploying a firewall: Is there a significant performance loss while incorporating a secure environment using a firewall for the internet connection? To what level of security should we expect without sacrificing the network performance? [14] The WAF functions by being positioned between the user-side client and the application server, thereby intercepting all data passed between the application server and the user. This traffic is examined by the WAF against various rules in an attempt to determine which data is valid and which is considered invalid

[13] and as discussed earlier that WAFs are also known as 'deep packet inspectors' because they will in detail check the packet is malicious or not. As discussed earlier, it is proved that WAFs are the best solution for securing the web server but the performance enhancement is WAF is very important factor.

The demand for computing power continues to increase in virtually every domain, from the basic desktop systems to the high-end computing platforms [15]. In the past, performance increase in processors was mainly reached by increasing clock frequency and designing more complex systems [16],[17]. The major focus is that how to place a WAF inside the environment to give maximum protection to the application but yet not compromising the performance of the application server. It is interesting to note that the intuitive belief about security to performance i.e. the more security would result in less performance, does not always hold in the firewall testing [14].

2.3 Parallel Computing

Parallel computing refers to usage of multiple number of processor in parallel to solve a problem instead of using single processor. [30]

The main goal which is desired from parallel computing is improvement in performance of the task assigned. Few years back people talked about parallelism in their research but now-a-days there is not a single company which is not offering multi-core systems for high computational tasks to increase their performance.

2.4 Moore's Law

Moore's law claims a long term trend in the constitution of computer hardware. He claimed that the number of transistors on an integrated circuit will get double approximately after every eighteen months of time. Moore's claimed this almost more than half century back and this trend is still followed by different computer hardware.

This claim shows that the capacity of the computer hardware is increasing

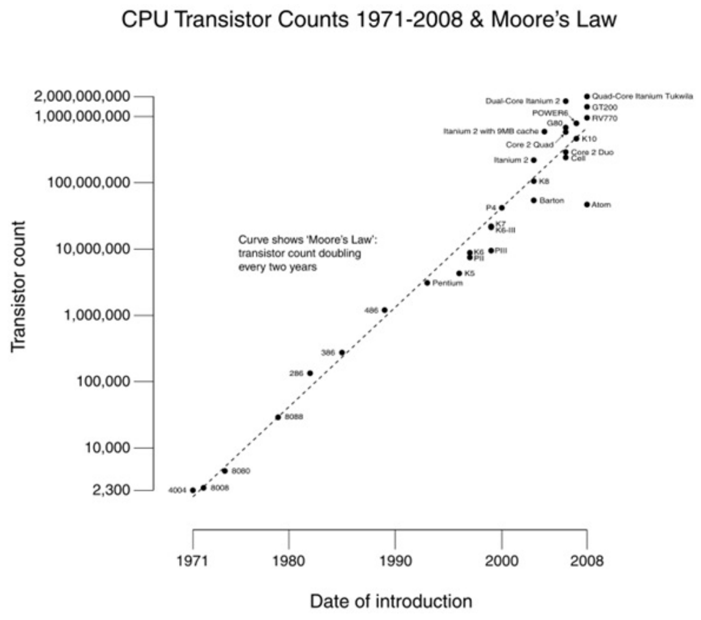


Figure 2.1: Moore's Law [24]

exponentially with time. Now the utilization of that capacity to its maximum level is the requirement of the era now. Utilizing hardware capacity to its maximum level by introducing different techniques and technologies can results in high performance systems.

2.5 Amdahl's Law

Amdahl's claimed in his law that, 'If in an application whose 50% of the code can be parallelized and 50% of the code cannot be parallelized run on infinite number of processors than the total time consumed by the application will cut down to half'.

The figure given below shows the comparison of parallelism of application and the number of cores on which application is running. This comparison was described by Amdahl in his law. Left side of the figure shows the performance benefits one can achieve by just doubling the number of cores on which the application is running without increasing the parallelism in the code. The Amdahl claimed that if time consuming by an application was 85% when it was running on single core than it will decreased to around 77.5 percent if that application start running on double number of cores.

The right side of the figure shows that after introducing the parallelism in the code by any means like multi threading, when application will run on single core take 85 percent time. This shows that if application has the capacity of parallelism but it is running on single core than it will not give any performance improvement.

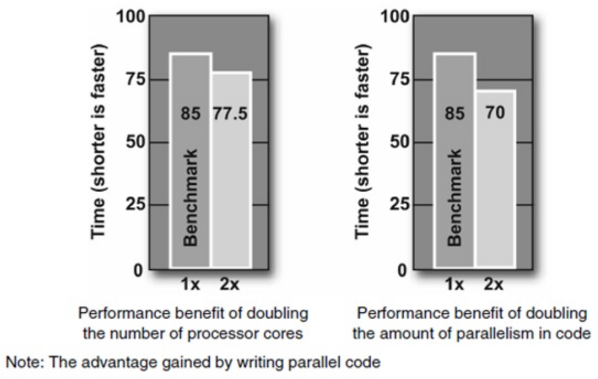


Figure 2.2: Amdahl's Law [16]

2.6 Parallelism saves power consumption

According to [30], the formula to calculate the power utilization in computing is given in figure 2.3

Where C is capacitance, V is voltage given to the chip and F is frequency.

$$\text{Power} = (C * V^2 * F)/4$$

Figure 2.3: Formula for Power consumption

Similarly according to [30], the formula to calculate performance of the application is given in figure 2.4

Here again F is frequency. If number of cores is increased to 2x in the

$$\text{Performance} = (\text{Cores} * F) * 1$$

Figure 2.4: Formula for performance evaluation

formula than the performance will also ideally go to 2x. Similarly if number of cores is increased to 2x but to maintain the performance to same level, frequency is decreased to *x. Exact same performance will be achieved at of the original power. (Decrease in Frequency also decreases the power require)

2.7 Revolution in computing

According to the figure given below, number of transistors on a chip or chip density is increasing continuously after every two years as stated by Moore's law. However, the increase in clock speed and the power offered are limited with time. Number of cores is increasing in hardware to increase these parameters now. Performance per clock is not increasing very much. Hardware is providing abilities but there is a great need of parallelism at software level. All major vendors are producing multi core systems now-a-days. Multithreaded application running over the hardware of this era is basically the requirement to gain high performance.

2.8 Overhead of parallelism

Parallelism brings some overheads along with more performance. There is a great need to overcome these overheads smartly to achieve desired performance boost. These overheads includes

- Cost of starting a thread or process

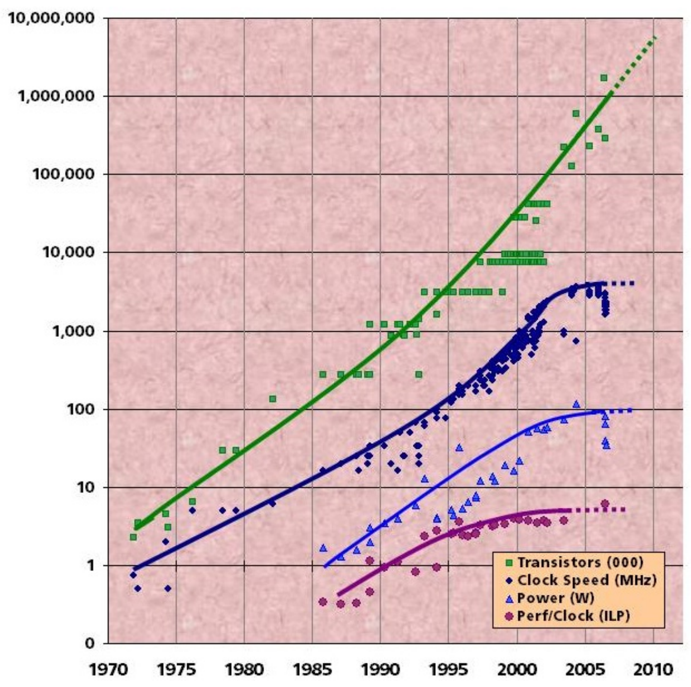


Figure 2.5: Evaluation in computing

- Cost of communicating shared data
- Cost of synchronizing
- Extra (redundant) computation

Each of these overheads may take milliseconds of time but the decision of units of work which run in parallel needs to be done very smartly so that performance is not affected by these overheads.

2.9 Relation of threads and cores

To find out the best combination of multi-threading over multi-core system I performed a small test. I wrote a multi-threaded program which was writing two files to the hard disk. I used different combination to check the performance of the program. The results I achieved are given in table 1.

Table 2.1: Threads vs. Cores

No of threads	No of cores	Time (sec)
1	1	60.50
1	2	61.27
2	1	58.47
2	2	37.88

The graphical representation is given in figure 2.6

The conclusion of this test is that a multi-threaded application running

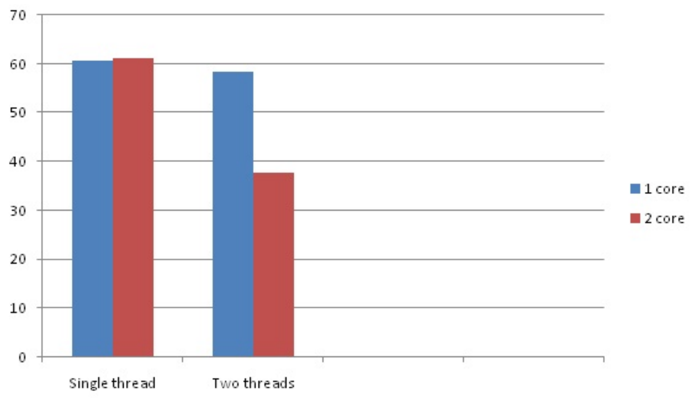


Figure 2.6: Threads vs. Cores

over multi-cores systems is key to achieve high performance solution. Prominent improvement can be noticed while running the last scenario.

2.10 Relevant Architectures

While doing the literature survey I went through different architectures which are somehow relevant to the architecture I am currently working on. I looked into the description of every architecture in detail and found out some pros and cons of the architectures too.

Michael R. Lyu and Lorrien K. Y. Lau [16] explored the firewall security and performance relationship for distributed systems. They performed experiments for different security levels of firewalls and quantify their performance results. Based on the test results, the impacts of the various firewall security levels on system performance with respect to transaction time and latency are measured and analyzed. They claimed that the belief that more security would result in less performance does not always holds true and they proved this claim via test results. The performance was evaluated via two performance indicators i.e. Latency and Total transaction time. They took results for both HTTP and FTP traffic.

Overall testing results in [16] showed that the firewall performance is affected only if the overhead incurred by the enhanced security control is significant when compared with the normal transaction time without the enhanced security control. If frequent connections with data of small size are required in communication, enhanced firewall security would be very likely to bring out some significant performance degradation to the private network. The con of the technique in paper [16] is that the authors have not talked about the parallelism at all. Secondly their firewall is Linux based network layer firewall and there are many issues with the firewalls at network layer as described in introduction session.

In paper [17] authors Errin W. Fulp and Ryan J. Farley introduced a firewall architecture that performs packet inspection under increasing traffic load, high traffic speeds and strict QoS. To enhance performance in this architecture authors used multiple instances of firewalls where each firewall was implementing a portion of security policy of organization. Packet was forwarded to every firewall and the gate as shown in figure 5. Each firewall process the packet based on their local security policy. If match found that is the packet is malicious, firewall will notify to the gate. As soon as match is found as any firewall packet will be declared malicious and will be discarded.

The advantage of this architecture is that as every firewall is involved in processing of each packet, the performance is improved regardless of the traffic load. Another advantage is in this architecture we can maintain the state. The disadvantages are that first of all this solution is not a scalable solution and secondly this is an expensive solution.

Another relevant architecture is proposed in paper [18] where authors talked about the three major issues when we talk about the firewall architecture.

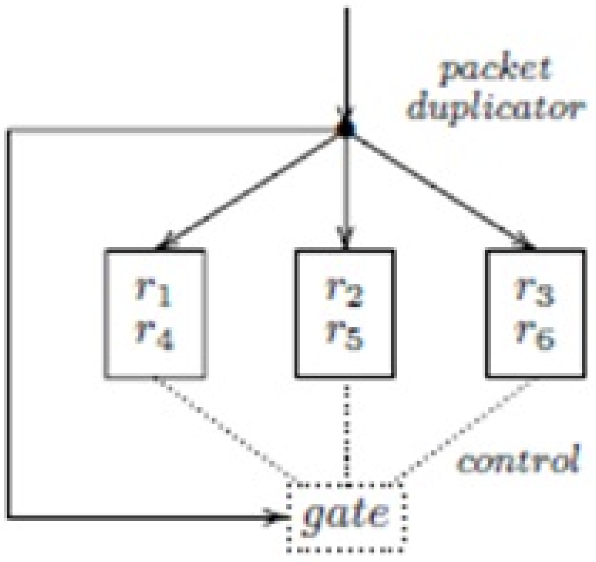


Figure 2.7: Function-parallel, rules distributed across an array of firewalls

First is the 'Performance issues' which will cause the reduction in the bandwidth throughput of the network. Second and third issues are 'availability' and 'complexity'. They said that among all these issues performance issue is the biggest because if that is not resolved the firewall may become bottleneck in the system. Authors discussed four different architectures of firewalls which are

- The Primary-Backup approach
- Multi-Primary Multipath Firewall Cluster
- Multi-Primary Firewall Cluster Sandwich
- Multi-Primary Hash-Based Stateful Firewall Cluster

Few of these architectures are for stateful firewalls and few are applicable to stateless firewalls. The most effective architecture which is also relevant to SWAF is 'Multi-Primary Hash-Based Stateful Firewall Cluster' because this architecture is for stateful firewalls like SWAF. This architecture is given in the figure below. In this architecture firewalls share the load without the load balancer because they are clones. All the instances of firewall receive same data all the time because a network layer hub with the port mirroring feature is used. Hash based approach is used on the firewalls to know that which firewall will filter the packet. The only advantage is this architecture is that it gives performance benefits. The disadvantages include complexity of architecture, costly solution and a problem that if one firewall fails during operation than a subset of packets won't get filtered.

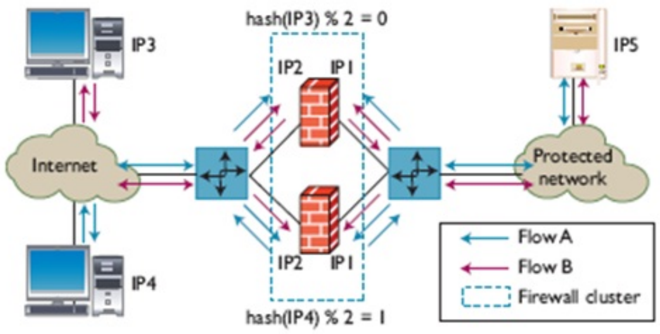


Figure 2.8: Multi-Primary Hash-Based Stateful Firewall Cluster

2.11 Conclusion

To conclude the discussion of this chapter I must say that there are different architectures which are proposed by different authors to improve the performance of a firewall but none of these architectures are focusing on multi-core platforms. Multi-core systems are very common now-a-days and utilizing the performance capacity of multi-core system can give the best results.

Chapter 3

Proposed System Architecture

3.1 Overview

In this chapter, I discussed the original SWAF (Semantic Based Web Application Firewall) architecture in little detail. There are few problems with the original architecture due to which the performance of current system is confined. However at higher loads clients normally do not wanted to compromise on performance of their applications because in e-commerce slow responses are highly discouraged. So to improve the performance over high loads a new architecture for SWAF is proposed in this chapter later on. Different challenges are discussed in the end which may affect the implementation of proposed architecture.

3.2 Current Architecture

The figure 3.1 contains the current architecture of SWAF. This figure shows different modules of SWAF.

The figure 3.2 shows the current architecture of SWAF in a bit detail. Here 'Listener' module is one separate thread which is always listening on a particular port. This means that listener module will receive requests on a defined port and will forward it to a thread from the thread pool every time.

The tasks of the modules in the next block i.e. starting from 'Interceptor' till 'Response generation' are executed by single thread which is highly in-efficient and independent way. The delay in any module of this block will

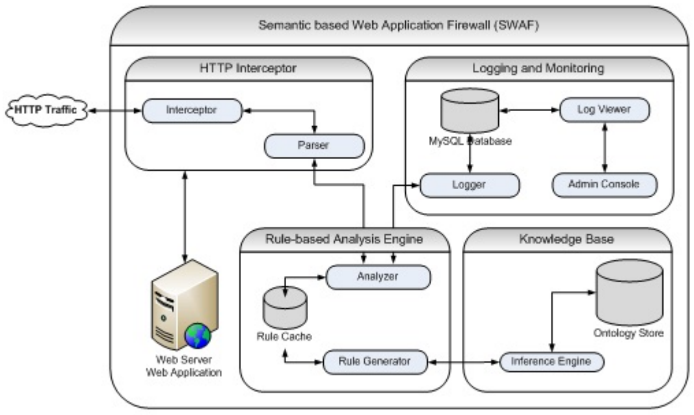


Figure 3.1: Current SWAF Architecture

hold the overall execution of the request. Size of thread pool defines that how much concurrent requests can be handled. This is a limitation of this architecture.

Current architecture of SWAF is not getting any benefit of multi-cores of the system because of the dependency of the biggest block of architecture.

Short description of each module of SWAF is given in figure 3.3

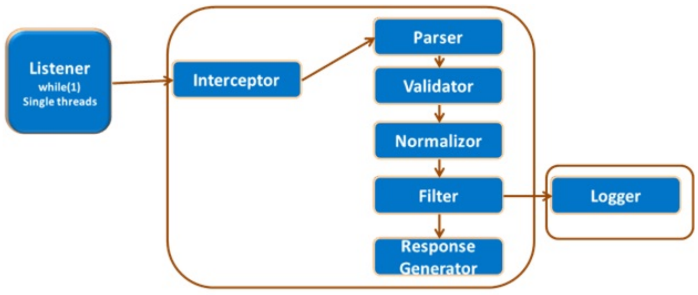
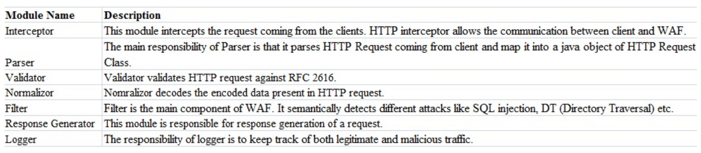


Figure 3.2: SWAF - In a bit detail



Module Name	Description
Interceptor	This module intercepts the request coming from the clients. HTTP interceptor allows the communication between client and WAF.
Parser	The main responsibility of Parser is that it parses HTTP Request coming from client and map it into a java object of HTTP Request Class.
Validator	Validator validates HTTP request against RFC 2616.
Normalizor	Nomralizor decodes the encoded data present in HTTP request.
Filter	Filter is the main component of WAF. It semantically detects different attacks like SQL injection, DT (Directory Traversal) etc.
Response Generator	This module is responsible for response generation of a request.
Logger	The responsibility of logger is to keep track of both legitimate and malicious traffic.

Figure 3.3: SWAF - Description of different modules

3.3 Weaknesses of existing solution

The weaknesses of current solution are given below.

- Single thread pool for each block
- More or less a serial architecture
- Almost no advantage of multi-cores

3.3.1 Single thread pool for each block

The two main blocks of this architecture contains separate thread pool. But the problem is that the thread pool for the main block (Starting from Interceptor till Response generator) is just one. The thread pool size of this block actually defines that how much number of requests can be handled concurrently.

3.3.2 More or less a serial architecture

Current architecture is not parallel architecture because modules are calling each other via the function calls. Function call is serial in nature because the caller of the function keeps on waiting for the result of the function it called. This actually means that interceptor module will keep on waiting till the response generator is not done with its work.

3.3.3 Almost no advantage of multi-cores

Now-a-days multi-core systems are very common. Multi-core systems help in enhancing the performance of the system if the architecture running over it supports it. The current architecture of SWAF does not help in increasing the performance of the system while running over multi-core system.

3.4 Problem Statement

Performance and security are desirable attributes for web servers. So to improve performance there is a need to design a parallelized architecture for SWAF on Multi-core systems. The theme is how to maximize the core utilization and minimize the communication of the sub-systems.

3.5 Proposed Architecture

The proposed architecture of SWAF contains independent working of major modules of SWAF. Each module performs its tasks and places the output in the memory buffers then a thread of next module starts working on that input given to it. Different modules are coordinating with each other via the buffers and not the function calls. Thread pool for different modules is separated from each other so that the working of one module may not affect other module.

Working of all modules is independent of each other and can be run on different cores of the system to enhance the performance of the overall system.

3.6 Challenges

The challenges I faced while working on this architecture includes

- Minimize System Complexity

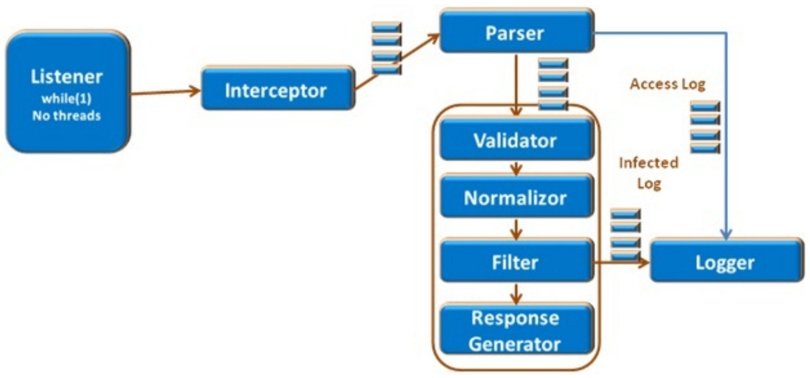


Figure 3.4: Proposed Architecture

- Minimize Inter-module Communication
- Reduce Sequential dependencies

3.6.1 Minimize System Complexity

System complexity is increased while we were trying to propose an architecture which could run on multi-core systems. So the first and the most important challenge was that the proposed architecture should not be that much complex that it may compromise the performance because of the complexity.

3.6.2 Minimize Inter-module Communication

By introducing the proposed architecture I am introducing enhanced communication between different modules. The second challenge I faced was that how can I optimize this communications so that the performance is not compromised the time taken by the inter modules communication.

3.6.3 Reduce Sequential dependencies

This architecture helped in reducing the sequential dependencies of different modules by introducing memory buffers between them to make them work independent of each other. There is a great need to learn that how to improve performance of the SWAF without increasing the complexity of the system.

3.7 Resolution of overheads of parallelism

As discussed in chapter 2, following are different overheads which can decrease the performance boost application can gain.

- Cost of starting a thread or process
- Cost of communicating shared data
- Cost of synchronizing
- Extra (redundant) computation

To resolve these overheads in proposed solution several measure are taken. I will briefly discuss them one by one.

3.7.1 Cost of starting a thread or process

In proposed multi-core architecture of SWAF, threads are not created each time we need them. Rather threads are one time created at start of the application and then they are saved in their corresponding thread pools. By this mechanism the time to create a thread each time is not required.

3.7.2 Cost of communicating shared data

In proposed multi-core architecture of SWAF, shared data is communicating between different modules by placing a user defined object in memory. Cost of sharing is reduced because the object is present in memory all the time and there is no I/O processing involved. However this cost is low but still present in proposed system.

3.7.3 Cost of synchronizing

Blocking Queue (Java defined Queue) is used for storage of data. Blocking queue is responsible for the synchronization of threads. It also solves the reader writer problem of threads. The details of blocking queue are given in chapter 4.

3.7.4 Extra (redundant) computation

There are no extra or redundant computations in SWAF. Each module works on its own tasks which are not shared by any other module of SWAF.

3.8 Conclusion

Protecting the application servers to the highest level possible without affecting the performance is desired by the industry. This is the era of multi-core systems and multi-core servers are present in almost all data centers now-a-days. However the capacity of a system cannot by fully utilized unless and until application running on that system is not capable of utilizing it. So the Multi-core and multi-threaded architecture is need of mission critical systems like SWAF.

Chapter 4

Implementation Details

4.1 Overview

This chapter contains implementation details of the proposed architecture. Different components of the system are discussed in detail in this chapter. Code snippets are also given with the components for the understanding of working. In the end responsibilities of different modules in the new architecture is discussed.

4.2 Implementation details

The main difference in the new architecture is that in this working of each module is separated with each other. Instead of function call different modules work independent of each other and places the results in a memory buffer after their processing. The next module picks up the object from the memory buffer for further processing. There are different components which are involved in the processing. These includes

- Stored data object
- Memory Buffer
- Threads Execution

4.2.1 Stored data object

Stored data object is custom object which is saved in the memory buffer by all modules after finishing their working. There are nine different parameter of HTTP Request which are saved in this data object. These includes

- Http Request
- Http Response
- Http Context
- Http Host
- Http Processor
- Http Request Executor
- Connection Reuse Strategy
- Http Request Handler Resolver
- Http Server Connection

The description of these parameters is given one by one to tell that what information is stored in all of these headers.

4.2.1.1 Http Request [28]

HTTP request parameter contains information about client while sending request message to server. HTTP Request includes first line of the message, the method to be applied to the resource, the identifier of the resource, and the Request header fields.

Request line contains method token, request URI and the information about the protocol version. The method token tells that what method should be applied to the resource identified by the request. Method token can have get, post, head, put, delete etc as its value in it. Request URI (Uniform Resource Identifier) tells the resource on which request should apply. Different options can be applied to Request URI. Asterisk '*' means that the request should not apply on any particular resource rather it should be applied to the server. Similarly absolute URI paths of resources are used in Request URI. An example of Request line is given below.

GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1

The identifier of resource helps in identifying the exact resource on which request needs to be applied along with request URI header. Identified is used in three different ways. If the request URI is absolute, identifier will be ignored. If the request URI is not absolute, the host is decided by this header. And if both the scenarios are not fulfilled then response must contain 'bad request' message in it.

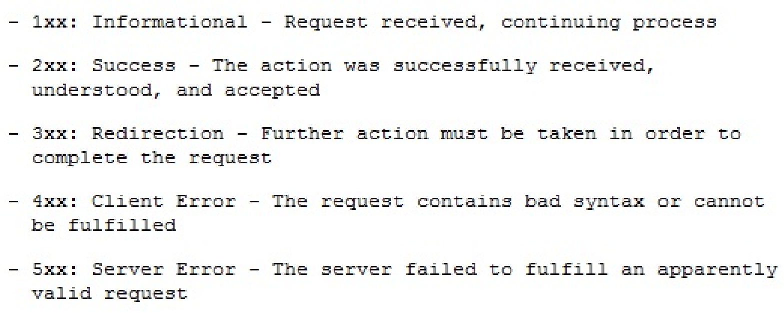
Request header fields contain different headers which give information of client to the server. These headers include Accept, Authorization, From, Host, If-match, and Range etc.

4.2.1.2 Http Response[29]

HTTP Response contains the server's response to the request. The first line of response contains protocol version, numeric status code and description of the status.

The status code is three digits code which tells the server's response and after that a short description is given by server to explain it a little. The first digit tells that from status code belongs to which class. Next two digits specify the exact status code.

The response header fields contain additional information from server to



```
- 1xx: Informational - Request received, continuing process

- 2xx: Success - The action was successfully received,
  understood, and accepted

- 3xx: Redirection - Further action must be taken in order to
  complete the request

- 4xx: Client Error - The request contains bad syntax or cannot
  be fulfilled

- 5xx: Server Error - The server failed to fulfill an apparently
  valid request
```

Figure 4.1: Response status codes

client which cannot be given in status line.

4.2.1.3 Http Context

HTTP context is a class in HTTP core library which encapsulates all HTTP-specific information about an individual HTTP request. This information is also saved in memory buffer so that client's complete information is preserved.

4.2.1.4 Http Host

HTTP Host is a class in HTTP core library that holds all of the variables needed to describe an HTTP connection to a host. This includes remote host name, port and scheme used. HTTP host helps in recognizing the client on the internet.

4.2.1.5 Http Processor

HTTP processor in an interface defined in HTTP core library. This is collection of interceptors which are responsible for different tasks. HTTP processor extends both HTTP Request Interceptor and HTTP Response Interceptor. Interceptors must be implemented as thread safe fashion.

4.2.1.6 Http Request Executor

It is HTTP protocol handler which implements all the requirements of the HTTP protocol for client side. It follows the requirements described by RFC 2616. HTTP Request Executor depends on HTTP processor to generate different headers.

4.2.1.7 Connection Reuse Strategy

Connection Reuse Strategy is an interface in HTTP core library. It decides that either connection can be reused or not for all further requests. On the decision of connection reuse strategy connection remains kept alive or discarded. Implementation of this interface must be thread safe.

4.2.1.8 Http Request Handler Resolver

Http Request Handler Resolver can be used to resolve an instance of matching a particular request URI. Usually the resolved request handler will be used to process the request with the specified request URI.

4.2.1.9 Http Server Connection

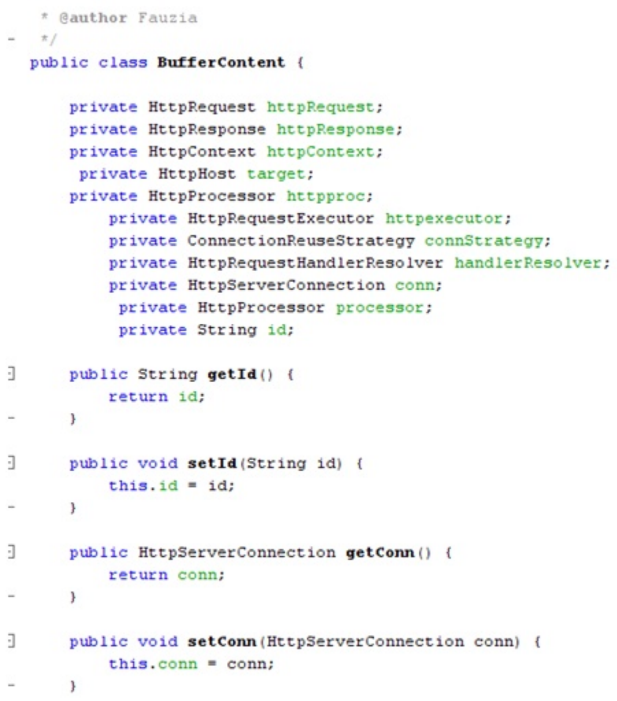
It is a server-side HTTP connection which can be used for receiving requests and sending responses.

In java code stored data object is an object of class 'BufferContent'. This class contains all the private data members of the classes described above. Getters and Setters are present in this class to access these private data objects. The visual description of class is given in figure 4.2

4.2.2 Memory Buffer

At first Java Queue was used in code for storing buffer content's objects in memory. But I was getting reader writer problem in that case. Reader writer problem is that if one thread is writing in queue the other thread is trying to read that content from the queue. To resolve that issue we used 'Blocking Queue' concept. This concept is present in java after Java 5. Blocking Queue's implementation is present in java.util.concurrent package.

Blocking Queue [25] resolves the reader writer problem at its own. The implementation characteristic of blocking queue includes following functionalities.



```
 * @author Fauzia
 */
public class BufferContent {

    private HttpRequest httpRequest;
    private HttpResponse httpResponse;
    private HttpContext httpContext;
     private HttpHost target;
    private HttpProcessor httpproc;
        private HttpRequestExecutor httpexecutor;
        private ConnectionReuseStrategy connStrategy;
        private HttpRequestHandlerResolver handlerResolver;
        private HttpServerConnection conn;
         private HttpProcessor processor;
         private String id;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public HttpServerConnection getConn() {
        return conn;
    }

    public void setConn(HttpServerConnection conn) {
        this.conn = conn;
    }
```

Figure 4.2: Buffer Content Class

- Blocks the thread when it tries to de-queue and queue is empty.
- Blocks the thread when it tries to en-queue and queue is full.
- If a thread tries to de-queue empty queue it is blocked unless data is inserted by any other thread.
- Thread trying to en-queue full queue is blocked unless some data is removed by any other thread.

There are different types of blocking queues [26] defined in java. These includes

- Array Blocking Queue
- Linked Blocking Queue

- Priority Blocking Queue
- Delay Queue
- Synchronous Queue

Short description of all these types is given.

4.2.2.1 Array Blocking Queue

In this type we need to tell the size of queue while declaring it at first. It behaves just like java arrays in which fixed size collection is created. This is FIFO (First In First Out) based collection.

4.2.2.2 Linked Blocking Queue

This type of queue is just like linked list in which no size needs to be specified. There is no maximum capacity of this queue theoretically. Practically limit of main memory is limit of this queue. This is also FIFO (First In First Out) based queue.

4.2.2.3 Priority Blocking Queue

In this type of queue priorities are assigned to objects stored in it. Objects with higher priority will get priority in processing.

4.2.2.4 Delay Queue

In this type of queues delay is assigned to data before appearing or disappearing from the queue. This kind of queue is normally used when such data items are present on which we wanted to apply such restrictions.

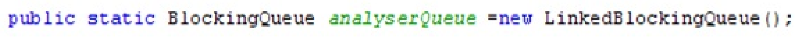
4.2.2.5 Synchronous Queue

In this type of queue, one thread always writes the item in the queue and another thread is removing item from the queue. There is no size limit in such queues.

I am using linked blocking queue in SWAF because of no size limit in it.

I will now discuss the working of queues in SWAF. For the discussion purpose I am using Analyzer module's queue. To declare a queue code line shown in figure 4.3 should be added.

To add object of buffer content code shown in figure 4.4 is added.



```
public static BlockingQueue analyserQueue =new LinkedBlockingQueue();
```

Figure 4.3: Defining analyzer queue

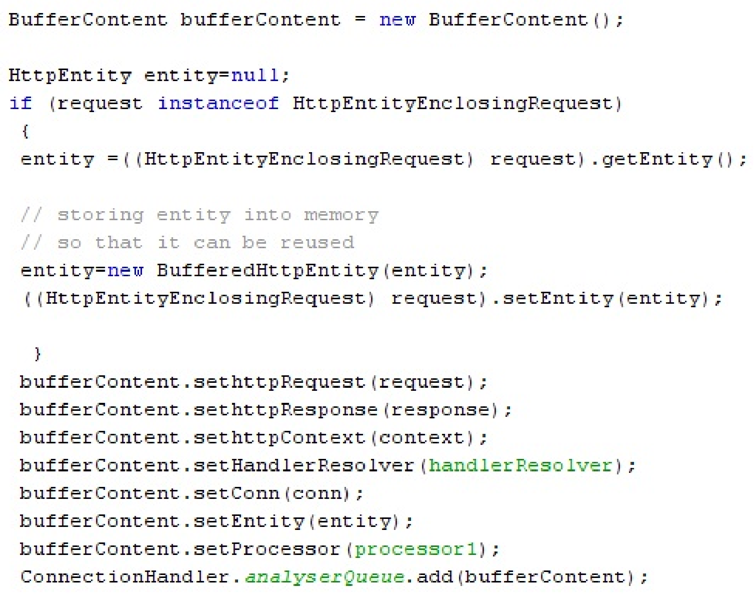
In the same manner object is retrieved from the queue by the next module.

4.2.3 Threads Execution

As discussed earlier, thread pools are separated for each module of SWAF. So to execute the working of a module, threads from its specific pool is executed. I am explaining this working with the help of Analyzer thread pool. Analyzer Invoker class contains definition of thread pools. Executor is defined in Analyzer Invoker as shown in figure 4.5

A fixed size thread pool is created for every module as shown in figure 4.6

To execute a thread of analyzer code shown in figure 4.7 is written.



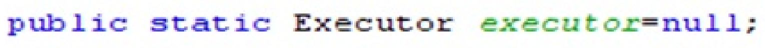
```
BufferContent bufferContent = new BufferContent();

HttpEntity entity=null;
if (request instanceof HttpEntityEnclosingRequest)
 {
 entity =((HttpEntityEnclosingRequest) request).getEntity();

 // storing entity into memory
 // so that it can be reused
 entity=new BufferedHttpEntity(entity);
 ((HttpEntityEnclosingRequest) request).setEntity(entity);

  }
 bufferContent.sethttpRequest(request);
 bufferContent.sethttpResponse(response);
 bufferContent.sethttpContext(context);
 bufferContent.setHandlerResolver(handlerResolver);
 bufferContent.setConn(conn);
 bufferContent.setEntity(entity);
 bufferContent.setProcessor(processor1);
 ConnectionHandler.analyserQueue.add(bufferContent);
```

Figure 4.4: Writing an object to queue



```
public static Executor executor=null;
```

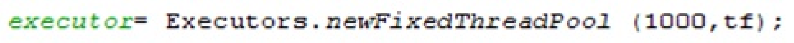
Figure 4.5: Defining executor

4.3 Responsibilities of Modules

After the implementation of new architecture, SWAF is mainly divided into four components. These includes

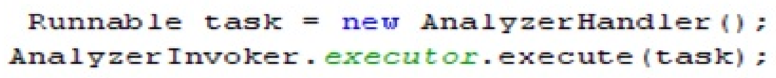
- HTTP Interceptor
- HTTP Parser
- Request and Response Filter
- Legitimate and Malicious Requests Logging

Brief description of all these modules is given below.



```
executor= Executors.newFixedThreadPool (1000,tf);
```

Figure 4.6: Fixed thread pool



```
 Runnable task = new AnalyzerHandler();
AnalyzerInvoker.executor.execute(task);
```

Figure 4.7: Invoking multiple threads

4.3.1 HTTP Interceptor

This module intercepts the request coming from the clients. HTTP interceptor allows the communication between client and SWAF.

4.3.2 HTTP Parser

The main responsibility of Parser is that it parses HTTP Request coming from client and map it into a java object of HTTP Request Class.

4.3.3 Request and Response Filter

Filter is the main component of SWAF. It semantically detects different attacks like SQL injection, DT (Directory Traversal), XSS (Cross Site Scripting), DoS (Denial of Service), DDoS (Distributed Denial of Service) etc.

4.3.4 Legitimate and Malicious Requests Logging

The responsibility of logger is to keep track of both legitimate and malicious traffic. It records the details of each request made to the application server.

4.4 Conclusion

Performance is very major concern of resource intense systems like SWAF. To enhance the performance we implemented a new architecture in which each module of SWAF is communicating via memory buffers instead of function calls. Concept of multi-threading is effectively used by assigned separate thread pools to each module. By running a multi-threaded system over multi-core systems we are expecting to have better performance.

Chapter 5

System Evaluation

5.1 Overview

In this chapter I will first discuss the evaluation criteria on which we are focusing while making the test runs and comparing them over different set ups. Afterwards I discussed different test runs I performed in both Windows machine and on high end Linux machines on different loads. I concluded the chapter in the end.

5.2 Evaluation Criteria

I used a tool named 'Jmeter' [27] by apache to test the system. Jmeter is stress testing tool which is used to measure the performance outcome of an application. In this tool there are several characteristics which help in setting up a test run for an application. Stress varies based on these characteristics defined in jmeter. The major characteristics of a test run includes following properties.

- Number of Requests / Second
- Total Number of samples in the test run
- Total Throughput of the test
- Error Rate

The brief description of all these properties is given below.

5.2.1 Number of Requests / Second

This parameter defines the concurrent number of requests generated by jmeter in that test run. Number of requests per second is defined as total number of users in the test run into the number of requests generated by one user in one second. This parameter defined the work load to the application in one second.

5.2.2 Total Number of samples in the test run

This parameter defines the total number of sample (Requests) made to the application by the test run. This parameter mainly depends on the number of samples recorded by jmeter during the recording phase of testing.

5.2.3 Total Throughput of the test

Throughput is defined rate of successful end to end communication. So in Jmeter all the requests having valid response add in to throughput parameter.

5.2.4 Error Rate

Error rate represents any type of error code in response message or delayed or no response from the application. The more the error rate the fewer throughputs will be generated by the test run. High error rate means application got down during the test run. This factor helps us determining the applications capacity.

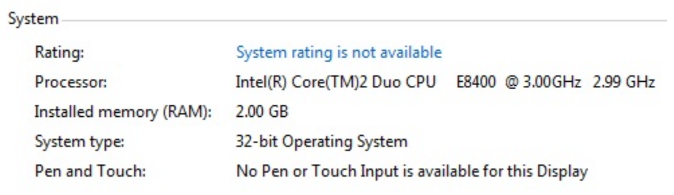
5.3 Testing Results

5.3.1 Testing Results on Windows Machine

First I performed a test with small number of sample size. Total sample size was around 2000 requests. I performed the test for different number of requests per second and ran it for both old SWAF code and for the new proposed SWAF's code. I plotted a graph to compare the results I collected from thesis test runs.

The system specification of machine on which SWAF application, web application (Web Goat in particular) and Jmeter was installed to test the systems is given in figure 5.1

Note that in the graph given below x-axis shows number of concurrent


System

Rating:	System rating is not available
Processor:	Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz 2.99 GHz
Installed memory (RAM):	2.00 GB
System type:	32-bit Operating System
Pen and Touch:	No Pen or Touch Input is available for this Display

Figure 5.1: Testing machine specifications

request which are generated by the system per second. This is not very huge test run so I used only 50 concurrent requests per second. Y-axis of graph shows the throughput the test run gave at the end.

In the figure it is very clear that at very small load like till nine requests per second performance of old SWAF is better than the proposed one. The reason is very obvious that the communication between the modules is increased in proposed SWAF's architecture. But on ten requests per second the throughput achieved by both systems are almost same. After that proposed system's performance is improving.

This shows that at low loads performance of the proposed architecture is not very good because of the increased communication between different modules.

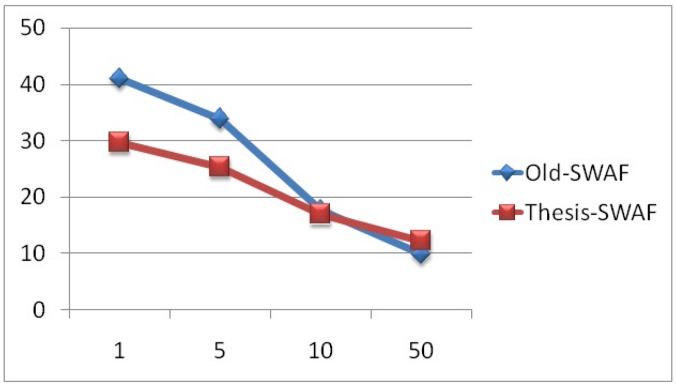
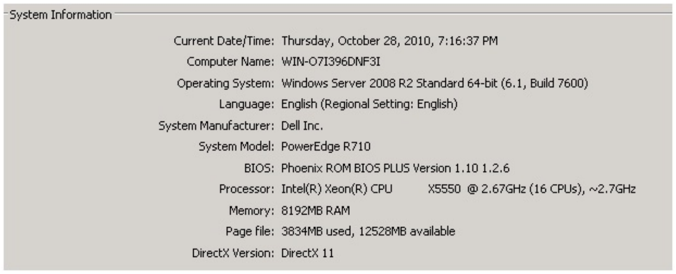


Figure 5.2: Comparison on low load

5.3.2 Initial Testing Results on High end Machine

I performed a test run with very huge number of data samples to test the performance of both systems. The high end server machine which was used to deploy SWAF, backend application (Web Goat in particular) and Jmeter machine had the following specifications.

Testing scenario included total sample size of more than 161,000 requests.

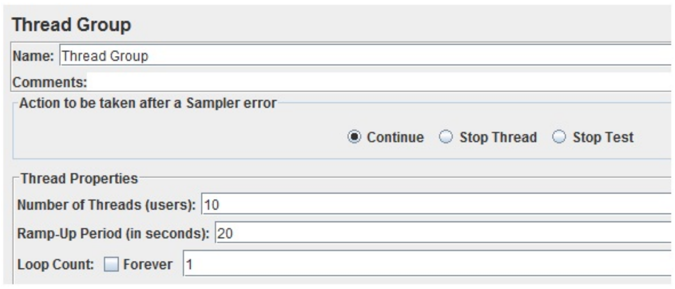


```
System Information
      Current Date/Time: Thursday, October 28, 2010, 7:16:37 PM
          Computer Name: WIN-O7I396DNF3I
       Operating System: Windows Server 2008 R2 Standard 64-bit (6.1, Build 7600)
               Language: English (Regional Setting: English)
    System Manufacturer: Dell Inc.
           System Model: PowerEdge R710
                   BIOS: Phoenix ROM BIOS PLUS Version 1.10 1.2.6
              Processor: Intel(R) Xeon(R) CPU        X5550  @ 2.67GHz (16 CPUs), ~2.7GHz
                 Memory: 8192MB RAM
              Page file: 3834MB used, 12528MB available
        DirectX Version: DirectX 11
```

Figure 5.3: System specification of high end machine

200 requests per second were generated by the jmeter to test the performance. Number of concurrent users was ten and they were generating 20 requests per second as shown in figure 5.3

This test was performed twice for old SWAF code. Old SWAF performs really well initially but after some time it slows down and later on the error rate increases very much because the SWAF become irresponsive. During first test run, SWAF performed really well till around 63,000 requests.



Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

[✓ Continue] Stop Thread Stop Test

Thread Properties

Number of Threads (users): 10

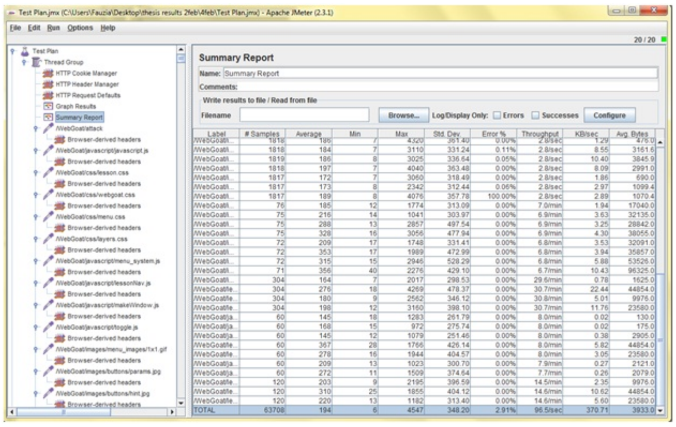
Ramp-Up Period (in seconds): 20

Loop Count: ☐ Forever 1

Figure 5.4: Number of requests per second

Throughput was around 95/sec at that particular time but after that SWAF stopped responding error rate continued to increase. I stopped the test run because error rate was increasing. The snapshot of intermediate result is given in figure 5.4

In second test run, I was bound to stop the test run after 140,000 requests



Test Plan.jmx (C:\Users\Fauzia\Desktop\thesis results 2feb\4feb\Test Plan.jmx) - Apache JMeter (2.3.1)

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

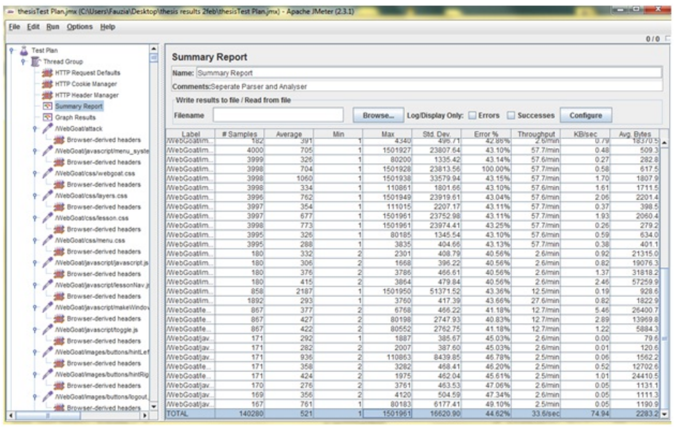
Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
/WebGoat/l...	1818	186	7	4320	361.40	0.00%	2.8/sec	1.29	476.0
/WebGoat/l...	1818	184	7	3110	331.24	0.11%	2.8/sec	8.55	3161.6
/WebGoat/l...	1819	186	8	3025	336.64	0.05%	2.8/sec	10.40	3845.9
/WebGoat/l...	1818	197	7	4040	363.48	0.00%	2.8/sec	8.09	2991.0
/WebGoat/l...	1817	172	7	3060	318.49	0.00%	2.8/sec	1.86	690.0
/WebGoat/l...	1817	173	8	2342	312.44	0.06%	2.8/sec	2.97	1099.4
/WebGoat/l...	1817	189	8	4076	357.78	100.00%	2.8/sec	2.89	1070.4
/WebGoat/l...	76	185	12	1774	313.09	0.00%	7.0/min	1.94	17040.0
/WebGoat/l...	75	216	14	1041	303.97	0.00%	6.9/min	3.63	32135.0
/WebGoat/l...	75	288	13	2857	497.54	0.00%	6.9/min	3.25	28842.0
/WebGoat/l...	75	328	16	3056	477.94	0.00%	6.9/min	4.30	38055.0
/WebGoat/l...	72	209	17	1748	331.41	0.00%	6.8/min	3.53	32091.0
/WebGoat/l...	72	353	17	1989	472.99	0.00%	6.8/min	3.94	35857.0
/WebGoat/l...	72	315	15	2946	528.29	0.00%	6.8/min	5.88	53526.0
/WebGoat/l...	71	356	40	2276	429.10	0.00%	6.7/min	10.43	96325.0
/WebGoat/l...	304	164	7	2017	298.53	0.00%	29.6/min	0.78	1625.0
/WebGoat/le...	304	276	18	4269	478.37	0.00%	30.7/min	22.44	44854.0
/WebGoat/le...	304	180	9	2562	346.12	0.00%	30.8/min	5.01	9976.0
/WebGoat/le...	304	198	12	3160	398.10	0.00%	30.7/min	11.76	23580.0
/WebGoat/ja...	60	145	18	1283	261.79	0.00%	8.0/min	0.02	130.0
/WebGoat/ja...	60	168	15	972	275.74	0.00%	8.0/min	0.02	175.0
/WebGoat/ja...	60	145	12	1079	251.46	0.00%	8.0/min	0.38	2905.0
/WebGoat/le...	60	367	28	1766	426.14	0.00%	8.0/min	5.82	44854.0
/WebGoat/le...	60	278	16	1944	404.57	0.00%	8.0/min	3.05	23580.0
/WebGoat/ja...	60	209	13	1023	300.70	0.00%	7.9/min	0.27	2121.0
/WebGoat/ja...	60	272	11	1509	374.64	0.00%	7.7/min	0.26	2079.0
/WebGoat/le...	120	203	9	2195	396.59	0.00%	14.5/min	2.35	9976.0
/WebGoat/le...	120	310	25	1855	404.12	0.00%	14.6/min	10.62	44854.0
/WebGoat/le...	120	220	13	1182	313.40	0.00%	14.6/min	5.60	23580.0
TOTAL	63708	194	6	4547	348.20	2.91%	96.5/sec	370.71	3933.0

Figure 5.5: SWAF test run I

because SWAF stopped responding and was continuously giving errors. The throughput at that time reached to 33.6 per second and error rate reached to 44.6 %. The snapshot of this test run of jemeter is given in figure 5.5

When this test was run for proposed SWAF architecture's implementation than the result was very consistent. The error rate was just 2.85 %. The consistent throughput in this case was 52.1/sec. The most important point for this test run is that SWAF was not irresponsive at any point of time.



Summary Report

Name: Summary Report

Comments: Seperate Parser and Analyser

Write results to file / Read from file

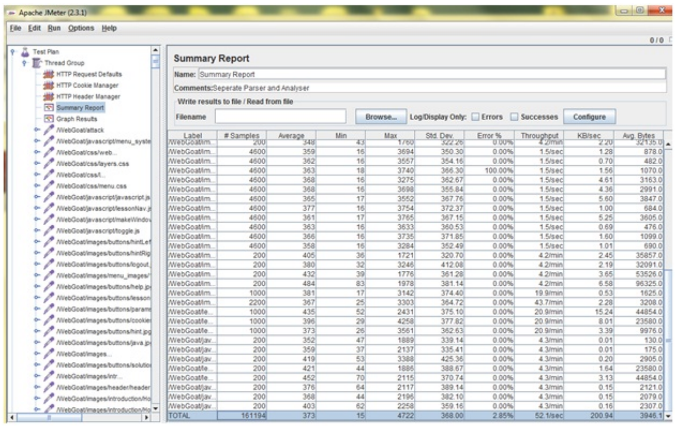
Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
/WebGoat/im...	182	391	1	4340	496.71	42.86%	2.6/min	0.79	18370.5
/WebGoat/im...	4000	705	1	1501927	23807.64	43.10%	57.7/min	0.48	509.3
/WebGoat/im...	3999	326	1	80200	1335.42	43.14%	57.6/min	0.27	282.8
/WebGoat/im...	3998	704	1	1501928	23813.56	100.00%	57.7/min	0.58	617.5
/WebGoat/im...	3998	1060	1	1501938	33579.94	43.15%	57.7/min	1.70	1807.9
/WebGoat/im...	3998	334	1	110861	1801.66	43.10%	57.6/min	1.61	1711.5
/WebGoat/im...	3996	762	1	1501949	23919.61	43.04%	57.6/min	2.06	2201.4
/WebGoat/im...	3997	354	1	111015	2207.17	43.11%	57.7/min	0.37	398.5
/WebGoat/im...	3997	677	1	1501961	23752.98	43.11%	57.7/min	1.93	2060.4
/WebGoat/im...	3998	773	1	1501961	23974.41	43.25%	57.7/min	0.26	279.2
/WebGoat/im...	3995	326	1	80185	1345.54	43.10%	57.6/min	0.59	634.0
/WebGoat/im...	3995	288	1	3835	404.66	43.13%	57.7/min	0.38	401.1
/WebGoat/im...	180	332	2	2301	408.79	40.56%	2.6/min	0.92	21315.0
/WebGoat/im...	180	306	2	1668	396.22	40.56%	2.6/min	0.82	19076.3
/WebGoat/im...	180	376	2	3786	466.61	40.56%	2.6/min	1.37	31818.2
/WebGoat/im...	180	415	2	3864	479.84	40.56%	2.6/min	2.46	57259.9
/WebGoat/im...	858	2187	1	1501950	51371.52	43.36%	12.5/min	0.19	928.6
/WebGoat/im...	1892	293	1	3760	417.39	43.66%	27.6/min	0.82	1822.9
/WebGoat/le...	867	377	2	6768	466.22	41.18%	12.7/min	5.46	26400.7
/WebGoat/le...	867	427	2	80198	2747.93	40.83%	12.7/min	2.89	13969.8
/WebGoat/le...	867	422	2	80552	2762.75	41.18%	12.7/min	1.22	5884.3
/WebGoat/jav...	171	292	1	1887	385.67	45.03%	2.6/min	0.00	79.6
/WebGoat/jav...	171	282	2	2007	387.60	45.03%	2.6/min	0.01	120.6
/WebGoat/jav...	171	936	2	110863	8439.85	46.78%	2.5/min	0.06	1562.2
/WebGoat/le...	171	358	2	3282	468.41	46.20%	2.5/min	0.52	12702.6
/WebGoat/le...	171	424	2	1975	462.04	45.61%	2.5/min	1.01	24410.5
/WebGoat/jav...	170	276	2	3761	463.53	47.06%	2.6/min	0.05	1131.1
/WebGoat/jav...	169	356	2	4120	504.59	47.34%	2.6/min	0.05	1111.3
/WebGoat/jav...	167	761	1	80183	6177.41	49.10%	2.5/min	0.05	1190.9
TOTAL	140280	521	1	1501961	16620.90	44.62%	33.6/sec	74.94	2283.2

Figure 5.6: SWAF test run II

Rather on high loads too SWAF behaved consistently.



Summary Report

Name: Summary Report

Comments: Seperate Parser and Analyser

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
/WebGoat/im...	200	348	43	1760	322.26	0.00%	4.2/min	2.20	32135.0
/WebGoat/im...	4600	359	16	3694	350.30	0.00%	1.5/sec	1.28	878.0
/WebGoat/im...	4600	362	16	3557	354.16	0.00%	1.5/sec	0.70	482.0
/WebGoat/im...	4600	363	18	3740	366.30	100.00%	1.5/sec	1.56	1070.0
/WebGoat/im...	4600	368	16	3275	362.67	0.00%	1.5/sec	4.61	3163.0
/WebGoat/im...	4600	368	16	3698	355.84	0.00%	1.5/sec	4.36	2991.0
/WebGoat/im...	4600	365	17	3552	367.76	0.00%	1.5/sec	5.60	3847.0
/WebGoat/im...	4600	377	16	3754	372.37	0.00%	1.5/sec	1.00	684.0
/WebGoat/im...	4600	361	17	3765	367.15	0.00%	1.5/sec	5.25	3605.0
/WebGoat/im...	4600	363	16	3633	360.53	0.00%	1.5/sec	0.69	476.0
/WebGoat/im...	4600	366	16	3735	371.85	0.00%	1.5/sec	1.60	1099.0
/WebGoat/im...	4600	358	16	3284	352.49	0.00%	1.5/sec	1.01	690.0
/WebGoat/im...	200	405	36	1721	320.70	0.00%	4.2/min	2.45	35857.0
/WebGoat/im...	200	380	32	3246	412.08	0.00%	4.2/min	2.19	32091.0
/WebGoat/im...	200	432	39	1776	361.28	0.00%	4.2/min	3.65	53526.0
/WebGoat/im...	200	484	83	1978	381.14	0.00%	4.2/min	6.58	96325.0
/WebGoat/im...	1000	381	17	3142	374.40	0.00%	19.9/min	0.53	1625.0
/WebGoat/im...	2200	367	25	3303	364.72	0.00%	43.7/min	2.28	3208.0
/WebGoat/le...	1000	435	52	2431	375.10	0.00%	20.9/min	15.24	44854.0
/WebGoat/le...	1000	396	29	4258	377.82	0.00%	20.9/min	8.01	23580.0
/WebGoat/le...	1000	373	26	3561	362.63	0.00%	20.9/min	3.39	9976.0
/WebGoat/jav...	200	352	47	1889	339.14	0.00%	4.3/min	0.01	130.0
/WebGoat/jav...	200	359	37	2137	335.41	0.00%	4.3/min	0.01	175.0
/WebGoat/jav...	200	419	53	3388	425.36	0.00%	4.3/min	0.20	2905.0
/WebGoat/le...	200	421	44	1886	388.67	0.00%	4.3/min	1.64	23580.0
/WebGoat/le...	200	452	70	2115	370.74	0.00%	4.3/min	3.13	44854.0
/WebGoat/jav...	200	376	64	2117	389.14	0.00%	4.3/min	0.15	2121.0
/WebGoat/jav...	200	368	44	2196	382.10	0.00%	4.3/min	0.15	2079.0
/WebGoat/jav...	200	403	62	2258	359.16	0.00%	4.3/min	0.16	2307.0
TOTAL	161194	373	15	4722	368.00	2.85%	52.1/sec	200.94	3946.1

Figure 5.7: Test run for Multi-core SWAF

5.3.3 Testing results on high end server after complete disintegration

I did a series of test runs on multi-core SWAF installed in VMware. Total eight cores and 3Gb RAM was assigned to the VM. Number of threads is given on x-axis whereas the throughput achieved is shown on the y-axis of the graph. The results are shown in the graph which shows that old SWAF performed better at low load but at high loads throughput gained by multi-core SWAF was much better.

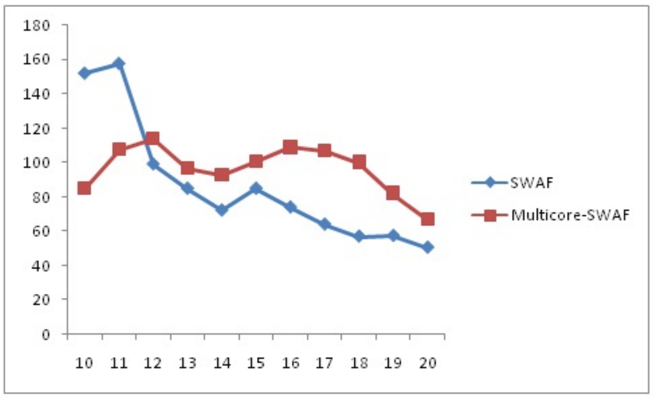


Figure 5.8: Comparison at different loads

5.4 Conclusion

Testing and evaluation is very important phase of system development. Open source stress testing software 'Jmeter' is used for testing of multi-core system. Jmeter is a product offered by Apache. Results are taken on both normal windows machine and high end linux server. It is proven that the result on high number of cores was really impressive.

Chapter 6

Conclusion and Future Works

6.1 Overview

This chapter contains the concluding remarks for the thesis. Importance of multi-threaded architecture is discussed and the advantages that multi-threaded application gains while running on multi-core system. Afterwards I discussed different aspects in which this system can be enhanced in future to increase the performance boost on high loads.

6.2 Conclusion

Multi-core systems are commonly used to deploy application servers and firewalls. Multi-core systems improve the performance of applications running over it. However application should be parallelized enough to utilize capacity offered by the multi-core systems. SWAF was previously designed to run on single core of the system. Different modules of SWAF were tightly coupled with each other and their working was highly dependent on each other. However, Proposed Multi-core architecture helped in reducing the response time required for each HTTP request by smartly sharing the load on different cores of the system. The research challenges I faced while designing multi-core architecture included reducing the proposed system's complexity, minimizing the increased inter-modular communication and most importantly optimized techniques are used to remove sequential dependencies among different modules.

In the proposed architecture of SWAF, all different modules are working independent of each other. Each module is having its own thread pool containing multiple numbers of threads which are responsible for performing tasks specific to that module. Each module of SWAF works independently and places the results in a memory buffer so that the other module can read it and perform further tasks on it. Java virtual machine is responsible to run each module on different core of the server machine. We have tested that JVM distributes the load to the available cores of the system. This architecture ensures better performance than the old architecture of SWAF over high loads as well. This claim is proved by system testing performed on high loads.

6.3 Future Works

Currently all the modules of SWAF are separated and are communicating via memory buffer. Different thread pools are created for each module of SWAF. Ideally each module is running on different cores of the system to increase the overall performance of SWAF. All the modules are currently running in single JVM (Java Virtual Machine) meaning by different parameters assigned to application running in JVM is shared by all modules. For example Java heap space assigned to JVM will be shared by all the modules. Heap space is the space given to JVM in memory of system.

For future, different modules can be made independent of each other by placing each module in separate JVM and communicating with each other. This may also increase the performance over high loads because all the components of SWAF will work totally independent of each other. Different modules will be loosely coupled with each other.

Another variation can be made by placing each module of SWAF on different system and communicating to each other via RMI (Remote Method Invocation). This variation will also increase the performance over high loads. The communication is also increasing in these two variations and it may be useful on really high loads.

Bibliography

[1] Abdul Razzaq, Ali Hur, Nasir Haider, Farooq Ahmad, “Multi-Layered Defense against Web Application Attacks” Information Technology: New Generations, 2009. ITNG ’09. Sixth International Conference on Digital Object Identifier: 10.1109/ITNG.2009.77 Publication Year: 2009 , Page(s): 492 - 497

[2] Tom Rowan, “Application firewalls: filling the void”, in Network Security Volume 2007, Issue 4, April 2007, Pages 4-7

[3] Open Web Application Security Project. “The ten most critical Web application security vulnerabilities”

[4] WebCohort, Inc., “Only 10% of Web applications are secured against common hacking techniques” http://www.imperva. com /company/news/2004-feb-02.html, 2004.

[5] G. Hulme. “New software may improve application security,” http://www.informationweek.com/story/ ,2001.

[6] Fulp, E.W.; Farley, R.J., “A Function-Parallel Architecture for High-Speed Firewalls” Communications, 2006. ICC ’06. IEEE International Conference on Volume: 5 Digital Object Identifier: 10.1109/ICC.2006.255099 Publication Year: 2006 , Page(s): 2213 - 2218

[7] E. D. Zwicky, S. Cooper, and D. B. Chapman, “Building Internet Firewalls”. O’Reilly, 2000.

[8] L. Qui, G. Varghese, and S. Suri, “Fast firewall implementations for software and hardware-based routers”, in Proceedings of ACM SIGMETRICS, June 2001.

[9] S. Suri and G. Varghese, “Packet filtering in high speed networks,” in Proceedings of the Symposium on Discrete Algorithms”, 1999, pp. 969 -970.

[10] R. L. Ziegler, “Linux Firewalls, 2nd ed. New Riders”, 2002.

[11] C. Benecke, “A parallel packet screen for high speed networks”, in Proceedings of the 15th Annual Computer Security Applications Conference, 1999.

[12] O. Paul and M. Laurent, “A full bandwidth ATM firewall”, in Proceedings of the 6th European Symposium on Research in Computer Security ESORICS’2000, 2000.

[13] Paul Byrne, “Application firewalls in a defense-in-depth design”, Network Security, Volume 2006, Issue 9, September 2006, Pages 9-11

[14] Lyu, M.R.; Lau, L.K.Y., ”Firewall security: policies, testing and performance evaluation”, Computer Software and Applications Conference, 2000. COMPSAC 2000. The 24th Annual International Digital Object Identifier: 10.1109/CMPSAC.2000.884700 Publication Year: 2000 , Page(s): 116 - 121

[15] Parallel Programming for Multicore, Electrical Engineering and Computer Sciences Department University of California, Berkeley http://www.cs.berkeley.edu/ yelick/cs194f07/

[16] Shameem Akhter, Jason Roberts, “Multi-Core Programming: Increasing Performance through Software Multi-threading”, Intel press 2006, http://www.intel.com/intelpress/samples/mcp_samplech01.pdf

[17] Fulp, E.W.; Farley, R.J., “A Function-Parallel Architecture for High-Speed Firewalls” Communications, 2006. ICC ’06. IEEE International Conference on Volume: 5 Digital Object Identifier: 10.1109/ICC.2006.255099 Publication Year: 2006 , Page(s): 2213 - 2218

[18] Pablo Neira Ayuso and Rafael M. Gaca, Laurent Lefevre; “Demystifying Cluster-Based Fault-Tolerant Firewalls” , Published by IEEE Computer Society Nov/Dec 2009

[19] Abdullah Kayi, Tarek El-Ghazawi, Gregory B. Newby; “Performance issues in emerging homogeneous multi-core architectures Simulation Modelling Practice and Theory”, Volume 17, Issue 9, October 2009, Pages 1485-1499

[20] S.R. Alam, R.F. Barrett, J.A. Kuehn, P.C. Roth, J.S. Vetter; “Characterization of scientific workloads on systems with multi-core processors”, in IISWC, IEEE,2006, pp. 225-236.

[21] S. Balakrishnan, R. Rajwar, M. Upton, K. Lai; "The impact of performance asymmetry in emerging multicore architectures", in ISCA'05: Proceedings of the 32nd Annual International Symposium on Computer Architecture, IEEE Computer Society, Washington, DC, USA, 2005, pp. 506-517. doi:¡http://dx.doi.org.proxygw.wrlc.org/10.1109/ISCA.2005.51¿. M.

[22] Chu, R. Ravindran, S. Mahlke; "Data access partitioning for fine-grain parallelism on multicore architectures", in MICRO'07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, Washington, DC, USA, 2007, pp. 369-380. doi:¡http://dx.doi.org.proxygw.wrlc.org/10.1109/MICRO.2007.11¿.

[23] P.F. Gorder, "Multicore processors for science and engineering", Computing in Science and Engineering 9 (2) (2007) 3-7. doi:¡http://dx.doi.org/10.1109/MCSE.2007.35¿.

[24] Moore's law, last modified on 24 March 2011, available at http://en.wikipedia.org/wiki/Moore

[25] Jakob Jenkov, 'Java Concurrency: Blocking Queues'; 2008 http://java.dzone.com/news/java-concurrency-blocking-queu

[26] R.J. Lorimer,'Concurrency: Blocking Queues and You'; 2004 http://www.javalobby.org/java/forums/m91820807.html

[27] Criteria Evaluated by Apache Jmeter (Stress Testing software, http://jakarta.apache.org/jmeter)

[28] part of Hypertext Transfer Protocol – HTTP/1.1 RFC 2616 Fielding, et al. available at http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html

[29] part of Hypertext Transfer Protocol – HTTP/1.1 RFC 2616 Fielding, et al. available at http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html

[30] Katherine Yelick, CS 194 Parallel Programming available on http://www.cs.berkeley.edu/ yelick/cs194f07