A cyber-security defense method using Docker containers

By

Weichen Wang

Thesis
Submitted to the Faculty of the
Graduate School of Some University
in partial fulfillment of the requirements
for the degree of

Master of Science

in

Computer Science

May, 2015

Nashville, TN

Approved：

Dr. Jules White

Dr. Aniruddha Gokhale

Acknowledgment

This thesis would not have been possible without the enthusiastic support and help of my advisor, Dr. Jules White, who guided me through this project. His rigorous attitude and helpful recommendations enabled me to overcome one difficulty after another, and finally made complete this research.

I also would like to thank my second thesis reader, Dr. Aniruddha Gokhale, for help reviewing and checking my thesis research. I also would like to thank Dr. Gokhale for his help in the Cloud Computing course, in particular, teaching me to keep a positive mental attitude when faced with difficulties in scientific research.

To all those who have discussed the topic with me, thank you. This is your work.

Finally, I would like to thank the EECS department at Vanderbilt. As an international student, you gave me the chance to study here, meet new friends, gain more knowledge, and broaden my horizons, I really appreciate this experience, thanks!

Table of Contents

List of Figures

Ⅰ. Introduction

Currently, millions of web applications are connected to the Internet and open for cyber-attack. Governments, militaries, corporations, financial institutions, hospitals and other entities collect, are processing and storing a larger and larger amount of confidential information on computers and transmitting that data across networks to other computers. At the same time, important data is vulnerable to growing and continual cyber-attacks, including denial-of-service attacks, SQL injection attacks, and many others.

In order to improve the security this critical data, researches have come up with a variety of different defenses, including newer hardware with built-in security features, higher-security network protocols, and expensive malware detection software. However, despite these advances, critical data still can be acquired through the Internet by attackers, especially when web applications are deployed in a cloud computing environment. Stopping these types of attacks and information capture is vital. Improper access to a database can cause the loss of control over thousands of patient records or critical intellectual property secrets. As companies store larger and larger amounts of data in the cloud, security will become even more important.

No organization is immune to these attacks – regardless of its size or sophistication. For example, the Cloud Security Alliance (CSA) reports that Amazon's retail website experienced a cross-site scripting attack in April 2010 that allowed the attackers to hijack customer credentials as they came into the site. In 2009, it reported, "numerous Amazon systems were hijacked to run Zeus botnet nodes."[1] And these are only two examples among hundreds of attacks that have been reported by CSA (cloud security alliance).

An emerging trend is to deploy web application cloud clusters, such as MongoDB, Hadoop, and Redis clusters. A key research question that we asked is if we cloud use the redundancy and connectivity in these clusters to devise a new defense mechanism? In particular, similar to a space-time tradeoff, can we sacrifice performance to get higher security by dynamically changing the cluster composition? Further, can we perform this type of tradeoff without

impacting the overall functionality of the cluster?

One possible approach to trade performance in a cluster for increased security is a moving target or hopping defense. A hopping defense is based on periodically moving application instances from one host to another, which may have different IP addresses or ports, so that attackers that gain access to a particular host can not hold onto control of the host for long periods of time. The approach gains security by limiting the time that any particular host can be compromised and under an attacker's control – thus reducing the damage that can be done.

However, there is a key concern with this method: we do not know how hopping will impact application performance and overall quality of service. A high hopping frequency may have an enormous adverse impact on performance, causing an unacceptable impact on quality of service. Further, we also do not know whether the method is feasible or will compromise the integrity of a clustered application and lead to the crash of the whole cluster.

In this paper, we set out to test the feasibility and performance of moving target defenses for clustered applications. One of the concerns with a hopping defense is the long startup times of virtual machines. In order to improve application instance startup time, we used container-based applications, which use Linux container mechanisms rather than virtual machines to isolate application instances. These container-based applications, such as Docker, can be started and stopped orders of magnitude faster than virtual machines. As part of the paper, we contrast the features of traditional virtual machine technology and the Docker container and show its advantages when deploying portable and light-weight application instances in a cloud cluster environment.

The remainder of this paper is organized as follow: Section Ⅱ describes our research motivation and background material, including the container technology we will analyze for a hopping defense in this paper; Section Ⅲ and Ⅳ cover how we built our test environment and how we chose web applications for our hopping experiments; Section Ⅴ discusses the

challenges we met in our experiments, and gives our solution to these challenges; Section Ⅵ presents empirical results from experiments with hopping defenses, and section Ⅷ presents concluding remarks and lessons learned.

Ⅱ. Background

Cyber-security is a top concern in nearly every major industry. Companies spend millions of dollars to build secure environments for their services. Unfortunately, despite the huge amounts of investment, they are still vulnerable to attack and routinely compromised. Hacking today is a very big issue and also a very big business. Hackers spend long, hard hours for payoffs and results that can easily reach into the millions. What are the security flaws that hackers exploit to attack? In a recent survey commissioned by Sungard Availability Services, the top 5 cyber-security threats to information systems were identified as [2]:

- No 1. vulnerable web applications (noted by 55% of respondents)
- No 2. being overall security “aware” (51%)
- No 3. out-of-date security patches (50%)
- No 4. failure to encrypt PCs and sensitive data (47%) No 5. obvious or missing passwords (44%)

This paper focuses on the No. 1 security threat, vulnerable web application, since defending web applications is a complex topic and less straightforward than the other flaws that are exploited. Designing a security awareness program, establishing a patch management schedule, encrypting PCs and sensitive data, and enforcing strong passwords are all relatively straightforward activities that do not need deep research. Securing vulnerable web applications, however, is a much more complex endeavor.

Today, more and more corporate services and business are built on cloud environments due to their elastic pricing and ease of management. In the remainder of this paper, we explore a new method to help increase security for these cloud-hosted web applications.

Cloud environments are highly dependent on virtualization. Virtualization technology separates a physical computing device into one or more “virtual” devices, each of which can be easily used and managed to perform computing tasks. However traditional virtual machine technology has some disadvantages when used for web applications. With traditional virtual machine technology, each VM includes not only the application, which is typically at most a few 100MBs in size, but also the entire virtualized operating system, which may be 10s of GB or more in size. Due to the large file sizes and limitations on network and disk bandwidth, it can be time and resource intensive to deploy a cluster based on virtual machine infrastructure. In order to solve this issue, a lightweight isolation technology - Docker - was developed.

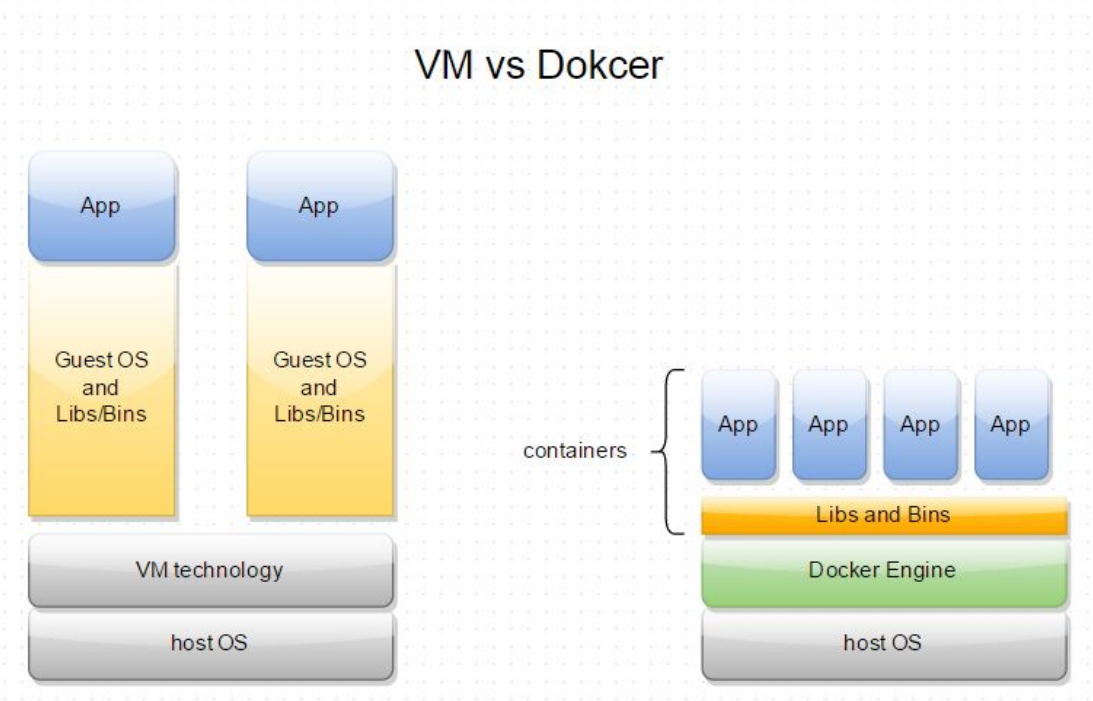

Fig 1. VMs vs Docker

Docker is an open source project that automates the deployment of applications inside software containers, which is a lightweight isolation mechanism for dividing up a single physical host into multiple virtual environments. By providing an additional layer of abstraction and automation of operating-system-level virtualization, the Docker Engine, on

Linux, it is possible to build much smaller application distribution units. Docker uses resource isolation features of the Linux kernel, such as cgroups and kernel namespaces to ensure the independence of each container, while avoiding the overhead of starting virtual machine.

The name Docker comes from the domain of ocean shipping. Just like containers on a cargo ship, a Docker container is an isolated packaging unit that can be easily moved. These standard isolation features are based on the isolation of different namespaces within the Linux kernel. The Linux kernel's support for namespaces completely isolates an application's view of the operating environment, including process trees, network, and user IDs. The kernel's cgroups also provide resource isolation that limits the usage of CPU, memory, block I/O, and network communication. At the same time, Linux chroot provides support for file system isolation. Docker also takes advantages of many other isolation tools with different tradeoffs, such as OpenVZ, systemd-nspawn, libvirt-lxc, libvirt-sandbox, qemu/kvm, to talk with Linux kernel. Docker has built its own driver API, libcontainer, to handle isolation issues [3]. These technologies, along with Docker’s APIs, allow applications to be packaged with only their dependent libraries and not an entire OS image, yielding much more space efficient distribution packages and faster startup times. Docker’s higher startup performance and small package size as opposed to traditional VMs, are the main reasons we chose Docker to do experiments in our paper. Both of these properties, startup time and package size, impact how quickly an application instance can be moved to a new host as part of a hopping defense.

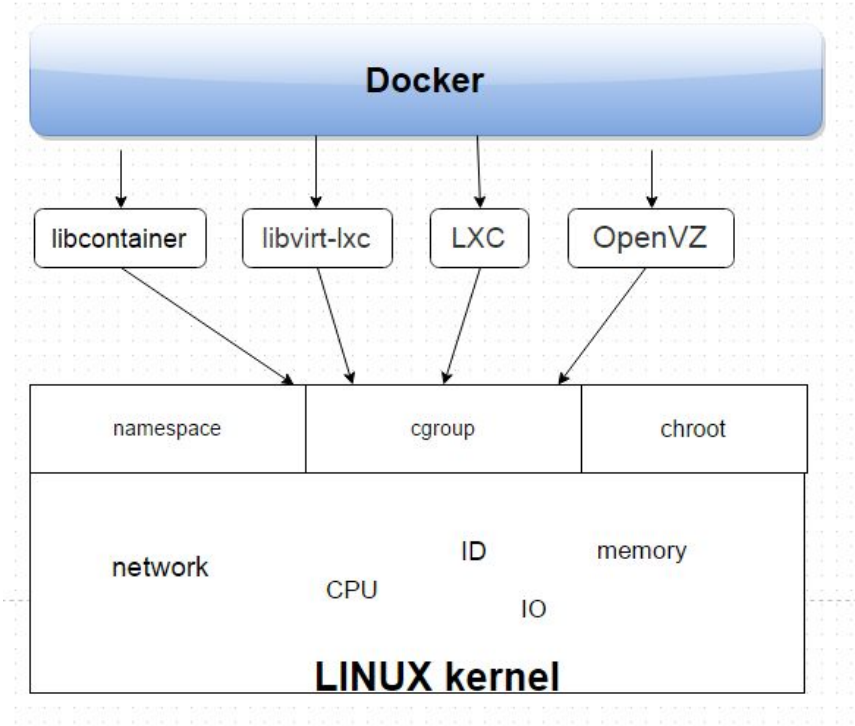

Fig 2. Docker API

Ⅲ. Research Questions

The fundamental assumption of this work is that, by hopping containers across hosts, we can improve security for web applications. A simple example will help to explain the idea. Assume that we have a database cluster with n containers, and a load balancer to distribute requests to a database across the containers. If a hacker gets access to container No. 1, he could steal data from this database instance as long as he has access to the container. By periodically hopping containers to another host, we can cut off this unauthorized access so that attackers only have limited time periods to exploit a compromised host. We can also change a subset of the configuration parameters during each move so that we can improve security by avoiding the reuse of a homogeneous environment configuration.

However, despite its potential, there are some key concerns with this method related to how periodic hopping will impact performance. To be more specific, there are two key research questions that need to be answered:

Research Question 1: How will increasing hopping frequencies impact performance?
Research Question 2: How will increasing the percentage of the total containers that are simultaneously hopped impact performance?

When we hop a container, we need to shut it down, and restart it again on another host. This process has the potential to produce significant disruption on the server-side, especially when we move multiple containers, because some application instances will be unavailable until they finish restarting. The research questions are focused on the two key parameters that will impact performance, how often each individual container is hopped and what percentage of the overall containers are simultaneously being hopped at any given instance in time. To answer these key research questions, we performed a number of empirical benchmarks.

IV. Experimental Setup

In order to accurately assess the impact of hopping on a real-world application, we built a web application with several database containers. Our hopping experiments were conducted using a MongoDB database cluster built from Docker container instances. The MongoDB database was sharded, meaning that its data was distributed and striped across the container isntances. In order to provide a single interface to the database, the cluster needs a load balancer to handle distribution and mapping of data to hosts. Our MongoDB Docker cluster was based on the best practices described in the article “Docker and MongoDB Sharded Cluster”, which provides a detailed roadmap to produce a sharded Mongo cluster [4].

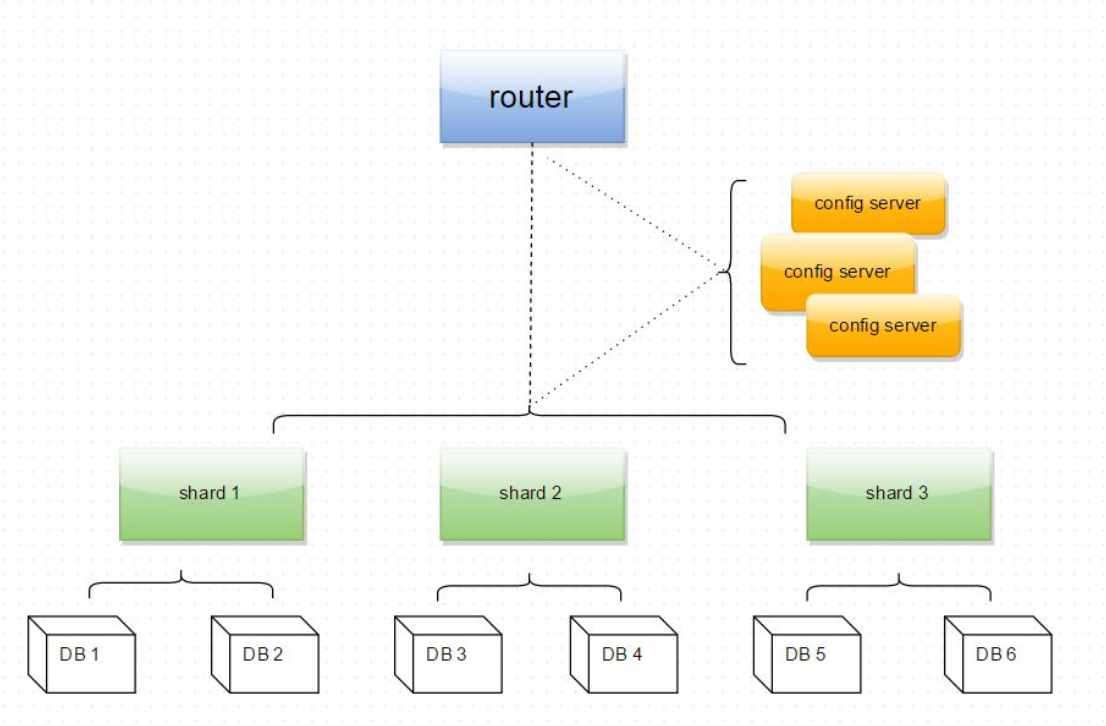

Fig 3. Mongo cluster using Docker containers

Figure 3 shows the structure of our 6 MongoDB container cluster. Although the primary database is sharded across 6 containers, we used 10 containers in total, including a Mongo router and 3 configuration server containers. We used this cluster setup to move the primary database containers at varying frequencies to see how performance was impacted.

Ⅴ. Experimentation Challenges

Experimentation Challenge 1: How to measure the impact of hopping on the performance of MongoDB. A key question was how to build a baseline for comparing performance and what performance metrics to emphasize. For example, we can test response time, latency, and many other parameters to look for deviations from the baseline, but we still need to choose which of these parameters are the best indicators of performance. Further, should the metrics be measured at the entire cluster-level or on individual application instances as they hopped?

There are additional considerations as well for our test environment because a database may have different kinds of requests, such as read and write requests that vary in complexity

and response time. For the experiments, we measured the throughput (requests/second) of write requests as the key performance metric, however we also recorded response time and other information to look for unexpected trends.

Experimentation Challenge 2: How to automate the hopping and performance analysis of the MongoDB cluster. Due to the consistency needed in hopping frequency and large loads required to accurately assess performance, manual experimentation methods were not feasible. We selected the four most popular open-source load testing tools: Grinder, Gatling, Tsung, and Apache JMeter for initial consideration as our experimentation automation infrastructure. All of the frameworks provided the key features that we needed, including automated statistical analysis of the results. However, one of the interesting discoveries that we made is that the overhead imposed by these tools was not consistent and that some of the tools were able to generate higher throughput and lower response times when tested against the same cluster. Figures 4 and 5 show comparative results for these four tools from a load test on the same cluster.

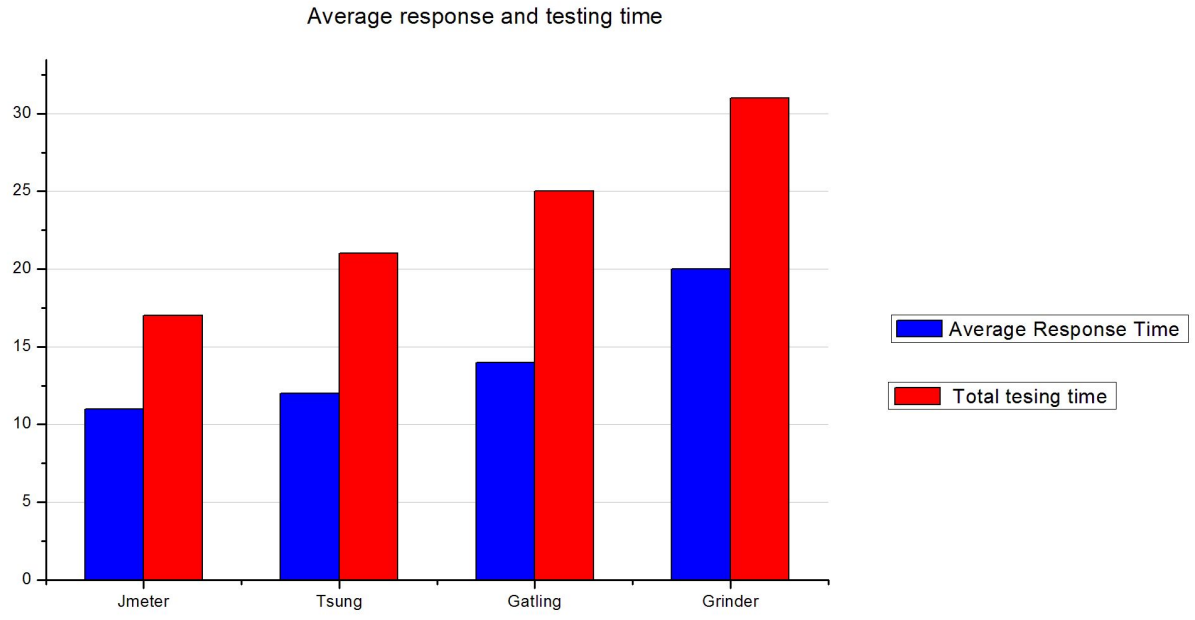

Fig 4. Loading test tools comparison of response & testing time

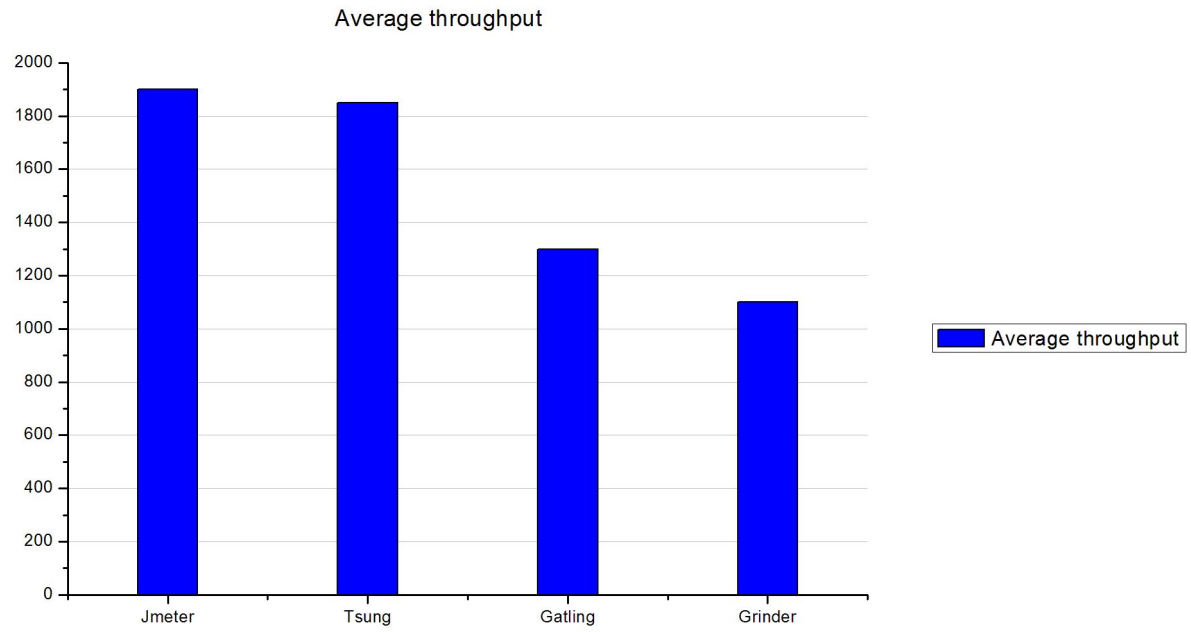

Fig 5. Loading test tools comparison of throughput achieved

As can be seen in the figures, Jmeter provided the best performance of the four tools. Further, Jmeter has a well-developed GUI, so we decided to use Jmeter for our experiments.

All of our experiments were performed on Amazon hosts with 64bit Ubuntu 14.04, 1024MB of RAM, and 2 CPU cores. We used the well-developed Mongodb Java API and Jmeter Java API to script our tests. We developed two Java functions for Jmeter to test the write and count throughput of MongoDB. The script for the write request generation for Apache JMeter is shown in Figure 6.

```
import com.mongodb.DB;
import org.apache.jmeter.protocol.mongodb.config.MongoDBHolder;
import com.mongodb.WriteResult;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCollection;
import com.mongodb.WriteConcern;
import com.mongodb.WriteResult;

// Get DB
com.mongodb.DB db = org.apache.jmeter.protocol.mongodb.config.MongoDBHolder.getDBFromSource("db", "test");

// Get collection to insert
DBCollection coll = db.getCollection("testCollection");
BasicDBObject doc = new BasicDBObject("name", "MongoDB").
                            append("type", "database").
                            append("count", 1).
                            append("info", new BasicDBObject("x", 203).append("y", 102));

// Insert object
WriteResult wr = coll.insert(doc, WriteConcern.ACKNOWLEDGED);

// Set response data
SampleResult.setResponseData(""+wr.toString(),"UTF-8");
```

Fig 6. Jmeter test script

Ⅵ. Empirical Results

A. Experiment 1: Establishing a Baseline for MongoDB Containerized Cluster Performance

The first experiment that we ran tested the performance of our MongoDB cluster built with Docker containers when no containers were being hopped. This initial experiment was performed in order to establish a baseline for performance. The results are shown in Figure 7.

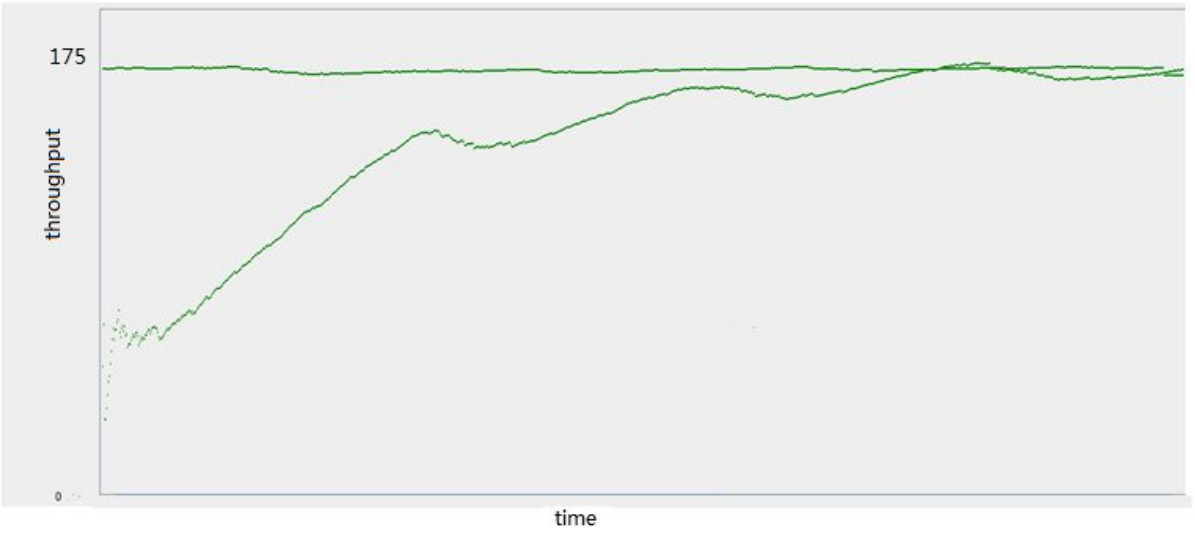

Fig 7. Jmeter baseline throughput analysis of the cluster

In this experiment part, we used 300 threads to generate requests and ramp up time of 2 seconds to start up the threads, which generates the initial upward curve in the graph. After the ramp up period, we can see the throughput stabilizes at roughly 175 transactions/second.

B. Experiment 2: Measuring the Impact of Hopping Frequency on Performance

This experiment was subdivided into five sub experiments. In each experiment, we hopped containers at varying frequencies to assess their impact on performance. For each experiment, we performed five trials and averaged the results. In each of the figures presenting the results, the x-axis represents the hopping interval in seconds, and y-axis is the throughput in transactions/second.

The first sub experiment moved a single container at varying hopping frequencies. Figure

8 show the results of hopping a single container at varying frequencies.

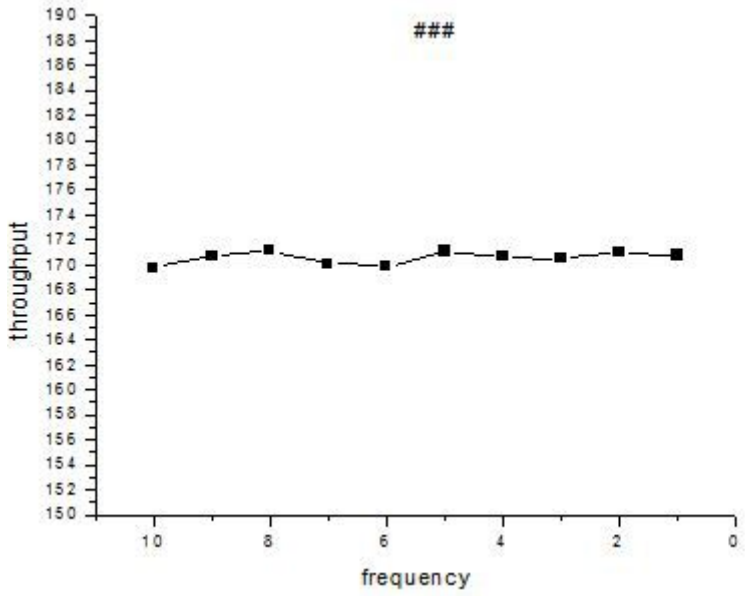

Fig 8. Single container hopping at varying frequencies

A surprising result shown in Figure 8 is that hopping a single container, even as fast as 1 hop per second, has little impact on the overall cluster performance. However, this result is due to the replica sets in MongoDB, which are a master-slave property to ensure that when one instance is down, another instance will take its place immediately, so that the throughput is not impacted significantly by hopping. It is therefore important to push our cluster further and continue testing to see the effect when we move more than one container at varying frequencies.

Figure 9 shows the performance when we moved two containers at different hopping frequencies.

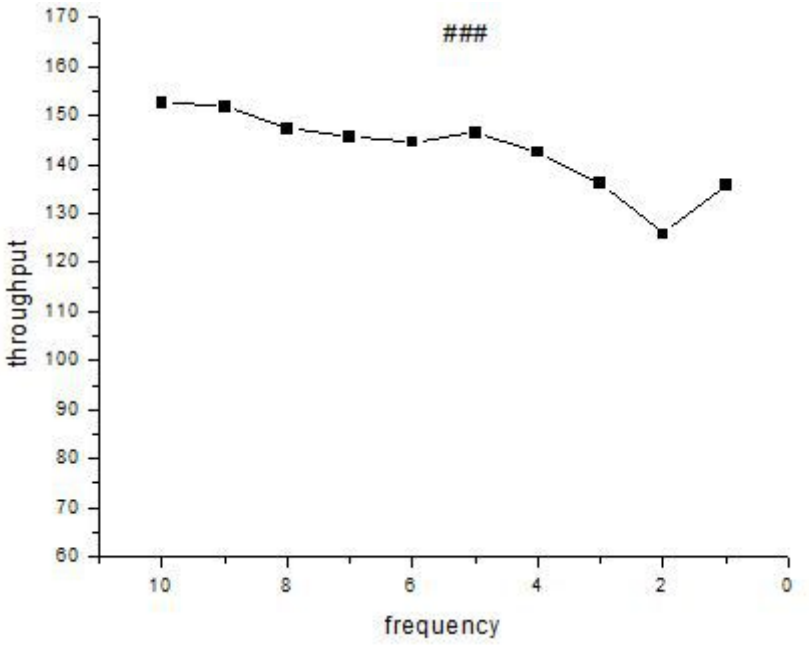

Fig 9. Two containers hopping at varying frequencies

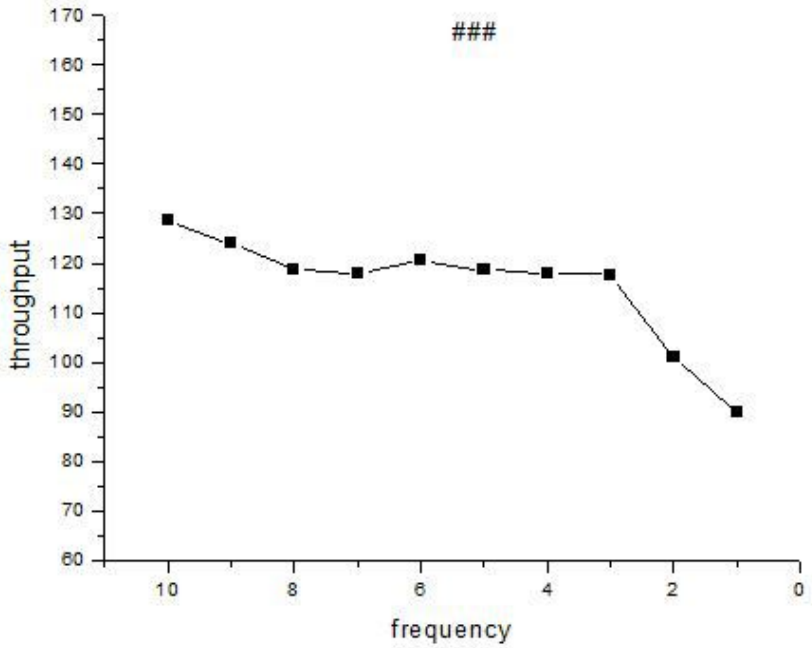

Fig 10. Three containers hopping at varying frequencies

In this diagram, we can clearly see that throughput goes down when we move more than one container at increasing frequencies. Figure 10 shows the performance of hopping three containers, or 50% of the entire cluster, at different hopping frequencies. There is almost the

same trend as when we move 2 containers, however, the throughput is more negatively impacted. As shown in Figures 10 and 11, hopping more than half of the containers in the cluster has a significant impact on performance, more than halving the peak performance achieved by the baseline cluster.

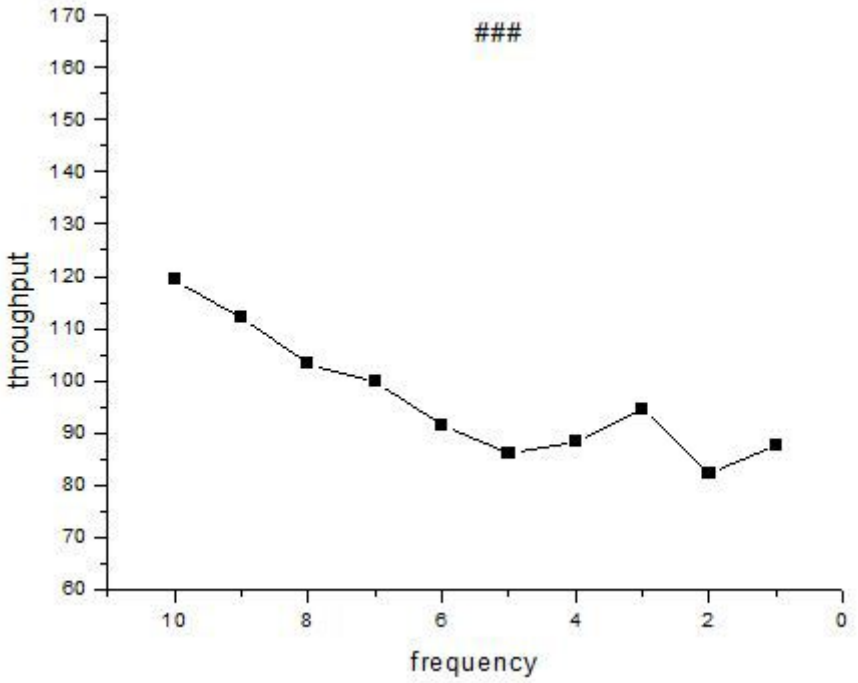

Fig 11. Four containers hopping at varying frequencies

According to the results shown in the figures, we find that throughput does go down linearly when we move containers more frequently. However, an important note is that we still achieve 50% of the performance of the baseline cluster even though up to 80% of the cluster nodes are being hopped at 1s intervals.

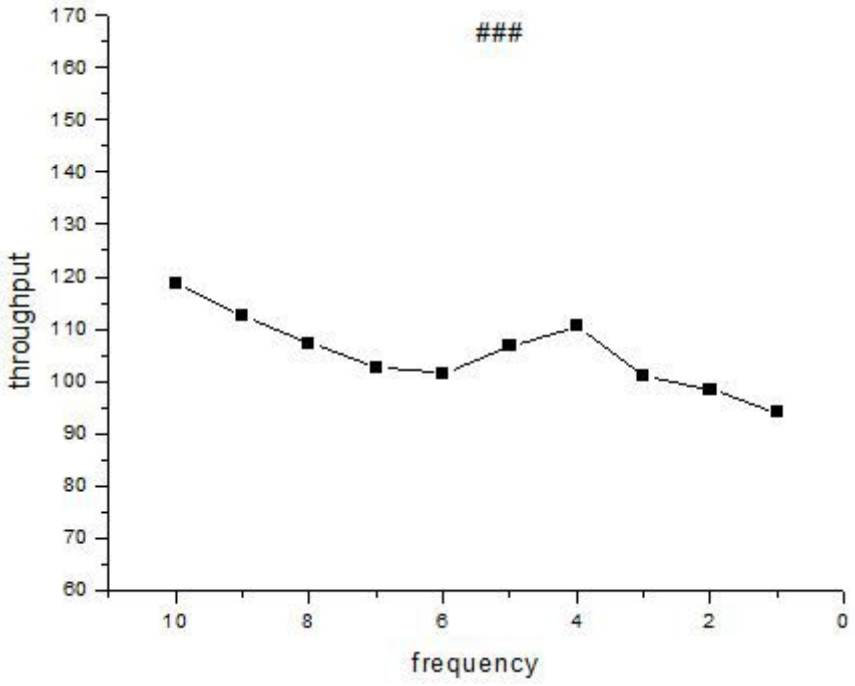

Fig 12. Five containers hopping at varying frequencies

C. Experiment 3: Measuring the Impact of Percentage of Containers Hopped on Performance

In this experiment, we fixed the hopping frequency and gathered data to analyze how the percentage of the overall cluster's containers that were hopped impacted performance. The results from the experiments are shown in Figures 13 and 14. From the two graphs, we find that, as expected, throughput goes down when we hop more containers but begins to level off when we move 50% or more of the containers.

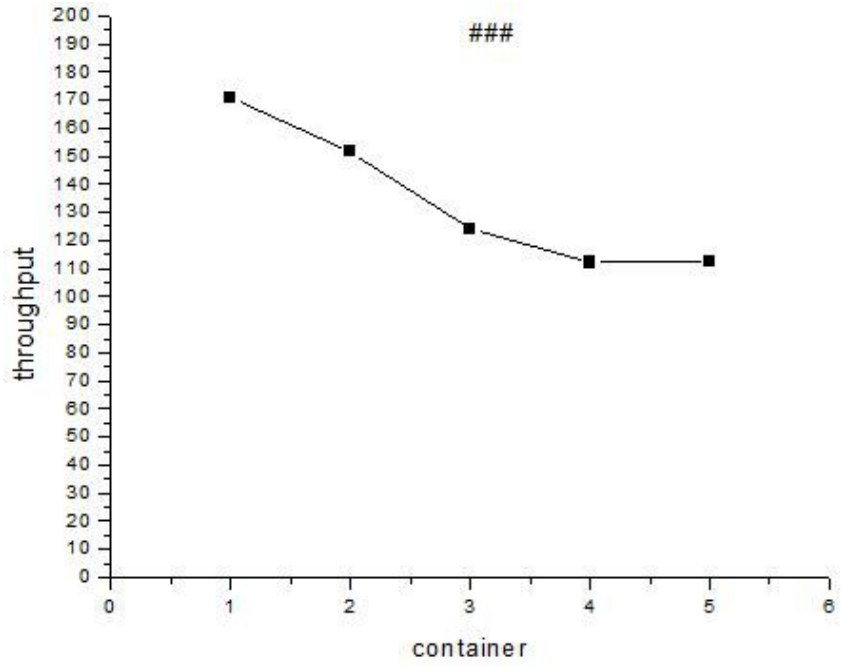

Fig 13. Impact of hopping varying numbers of containers at a 9s interval

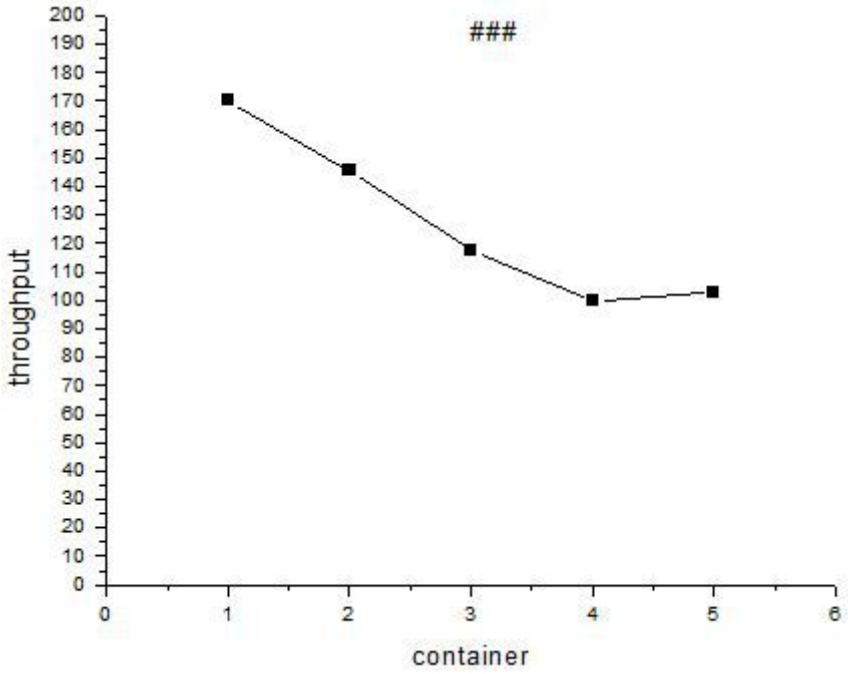

Fig 14. Impact of hopping varying numbers of containers at a 7s interval

D. Analysis of results

A key conclusion that we can draw from the results is that container hopping is a feasible cyber-security defense mechanism. However, as shown in the results:

1) The throughput will decrease with increases in hopping frequency
2) The throughput will decrease as larger percentages of the cluster's containers are simultaneously hopped

A shortcoming of the experiments is that our sample size is still not large enough to draw strong conclusions and more research is still needed. However, the initial results for container hopping are promising. In future work, we need to automate and generate much larger test cases.

E. Future work and additional experiment for some other application cluster.

The primary area for improvement is more robust experimentation and expansion of the sample data. In future work, not only do we need to do more experiments for MongoDB clusters with different hopping frequencies, but also with different CPU and memory resources, which can impact container startup time and change the results. With enough data, we also believe it is possible to build a linear model to predict how container hopping parameters impact performance metrics, of the form:

$$T =< f,\ \mathrm{p}, \mathrm{C}, \mathrm{M} >$$

where T is throughput, f is hopping frequency, p is the hopping container percentage, C is the CPU parameter, and M is memory. With this type of model, it will be more possible to design accurate hopping strategies according to meet specific performance requirements.

Another area for improvement is to test this model on other web applications. Success on MongoDB clusters does not ensure success on other applications. We tried to build a Redis cluster using Docker containers, however Jmeter can not log into the Redis container to execute read and write operations, so the experiment failed. In future work, we need to find a solution to fix this issue.

Ⅶ. Related work

Cyber security is a subject, which can be investigated from many different angles, and researchers have produced a variety of interesting ideas in this space. A number of prior works have looked at hopping as both an attack and defense mechanism for electronic signals. The most well-known attack research in this area is the Virtual Lan hopping attack, Steve A. Rouiller's work [5] describes this attack scenario. In addition to hopping attacks, some research has also used hopping as a defense. Sharma et al. investigated senser security using hopping in 2010 [6]. Lee et al. have also investigated port hopping to build resilient networks [7].

Large-scale clusters are increasingly being used to build enterprise services and have consequently received significant research attention. Yurcik et al. describe clustering techniques and challenges in enterprise environments [8]. Pourzandi et al. have also specifically investigated cluster security [9]. Hwang et al. have looked at approaches for improving cluster security specifically in cloud environments [10]. We chose Docker as the experimental platform because of its improvements to application deployment and instance startup. However, since Docker is new, its overall security is still being investigated. Walsh et al. have specifically found a number of potential Docker security issues [11].

Ⅷ. Concluding remarks and lessons learned

In this thesis, we investigated a new security mechanism for cluster-based applications: container hopping. In order to assess the viability of this defense mechanism, we used a new virtual machine technology, Docker, to build a MongoDB database cluster and assess the impact of container hopping on overall performance. One initial hope of the research was to perform a linear regression to build a model of the relationship between throughput and hopping frequency, and throughput and hopping container percentage. However, we found that the small sample size in our experiments generated models that poorly fit the data. Despite the inability to create a model of these relationships, we still learned some important lessons:

1) When we move just one container, even at frequencies as high as one hop per second, there was little impact on the throughput of the cluster.
2) Even when we moved half of the containers, the throughput was still up to 70% of the original cluster with only a 30% increase in response time.
3) When we moved almost all of the containers (5 containers out of 6 in our test), the throughput was still around 50% of the original cluster, but the response time had a huge increase, which in some cases doubled.
4) When the hopping interval was very large, more than 20 minutes, the performance had little to no change. The reason for this is the fast start up and tear down of Docker containers, which is just 1 or 3 seconds for each container, allows the cluster to return to its original state very quickly after a hop.
5) If your server is under frequent attack, you can easily hop 10% of your Docker containers at a 10s frequency with 5% or less in performance loss.

Overall, the results warrant future research into container hopping and its effectiveness as a cyber-security defense mechanism.

References

1) CSA report, The Notorious Nine Cloud Computing Top Threats in 2013. Online Available: https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf

2) Cyber Security Threat To Information Systems Today
Online available: http://blog.sungardas.com/#sthash.VJIb5WC1.dpbs

3) Introducing execution drivers and libcontainer. Online available:
http://blog.Docker.com/2014/03/Docker-0-9-introducing-execution-drivers-and-libcontainer/

4) Docker and MongoDB Sharded Cluster. Online available:
https://sebastianvoss.com/Docker-mongodb-sharded-cluster.html

5) Virtual LAN Security: weaknesses and countermeasures. Online available: http://www.sans.org/reading-room/whitepapers/networkdevs/virtual-lan-security-weaknesses-countermeasures-1090

6) Gaurav Sharma. Security in Wireless Sensor Networks using Frequency Hopping[J]. International Journal of Computer Applications (0975 – 8887), Volume 12 – No.6, December 2010

7) Henry C.J. Lee, Vrizlynn L.L. Port Hopping for Resilient Networks. VTC2004-Fall. 2004 IEEE 60th, 2004

8) William Yurcik, Gregory A. Koenig. Cluster Security as a Unique Problem with Emergent Properties: Issues and Techniques. 5th LCI International Conference on Linux Clusters 2004

9) Pourzandi, M. Clusters and security: distributed security for distributed systems. Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on (Volume:1) 2005

10) Kai Hwang. Cloud Security with Virtualized Defense and Reputation-Based Trust Mangement. Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference 2009

11) Daniel Walsh. Are Docker containers really secure?
Online available: http://opensource.com/business/14/7/Docker-security-selinux

12) Lenny Zeltser. Security Risks and Benefits of Docker Application Containers. Online available: https://zeltser.com/security-risks-and-benefits-of-Docker-application/

13) NoSQL database mongodb explained. Online available: http://www.mongodb.com/nosql-explained

14) Aaron Schram. MySQL to NoSQL: data modeling challenges in supporting scalability. Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity Pages 191-202, 2012.

15) Ryan Chamberlain. Using Docker to support reproducible research.

16) Hao Zhang , Bogdan Marius Tudor. Efficient In-memory Data Management: An Analysis.