The University of Hong Kong

Faculty of Engineering

Department of Computer Science

COMP7704
Dissertation Title
Building Reliable and Secure Replication Service in Distributed Systems

Submitted in partial fulfillment of the requirements for the admission to the degree of Master of Science in Computer Science

By
Yi Ning
HKU student no. 3035249186

Superviosr: Dr. Heming Cui
Date of submission: 01/11/2016

Abstract

By allowing applications to survive hardware failure, state machine replication (SMR) technique is widely used in computing environments. Meanwhile, the traditional SMR protocol like Paxos can hardly serve as an efficient replication protocol in distributed systems because its consensus messages go through TCP/IP layer, causing the consensus latency to be very big. The main target of this dissertation is to help build FALCON, a reliable and secure Paxos protocol with the RDMA device, and imbed this RDMA-based Paxos protocol into the databases and virtual machines.

Our group have implemented a scalable RDMA-based Paxos protocol for providing fault-tolerant applications, based on the approach of replicating the execution of the leader application. By imbedding this protocol into 9 widely used, unmodified server programs, we found the overhead of FALCON is low even in the local area network. I have imbedded FALCON into KVM (Kernel-based Virtual Machine) for providing high availability virtual machines. The critical applications running in the virtual machines can be guaranteed available all the time even in the case of hardware failure or host operating system serious fault. Evaluation on three widely used open-sourced databases (e.g. MySQL and MongoDB) shows that this solution has moderate overhead.

Building Reliable and Secure Replication Service in Distributed Systems

by

YI Ning

A dissertation submitted
In partial fulfilment of the requirement for
the degree of Master of Science
at the University of Hong Kong

October 2016

Declarations

I declare that this thesis represents my own work, expect where due acknowledgment is made, and that it has not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

Acknowledgements

First and foremost, I would like to take this opportunity to express my deep sense of gratitude to all the people who gave advices and encouragements to me during my study in HKU.

I am very much thankful to my supervisor, **Dr. Heming Cui**, for his continuous support, valuable guidance and keen encouragement throughout my studies. Particularly, I would like to thank him for giving me precious comments and suggestions on my dissertation. He is a responsible supervisor who is always strict with my dissertation. He not only offers advices to me, but also gives me lots of inspiration on my life.

I would like to thank the members in our System Group, **Cheng Wang, Jianyu Jiang, Zhiyang Li and Jingyu Yang**, for their encouragement and advices. **Cheng** is my senior and often gives me a hand. He is the person who implements the FALCON and helps me imbed FALCON into KVM. **Jingyu Yang** is a seasoned researcher and I always seek his help in life.

A debt of gratitude is also owed to the people in Main Building Office 110 and 112. With the accompany of them, the study life seems relaxing and interesting. They provide me a comfortable working environment during my study. Communicating with them in my leisure time made me feel happy.

Last but not the least, I would like to thank my close friend, **Shuangxi Wang**, for his encouragement on my study and life. He is an experienced senior specializing in distributed systems for many years. We shared the happiness and sadness with

each other. Once I come across some problems in study, I would immediately turn to him for help. He is so selfless that never refuses my request. Thank you.

Contents

Abstract

Declarations

Acknowledgements

1. Introduction

    1.1 sate machine replication and paxos

    1.2 limitations of traditional paxos

    1.3 remote direct memory access

    1.4 purpose of dissertation

    1.5 organization of the dissertation

2. review of literature

    2.1 database replication

        2.1.1 redis replication

        2.1.2 mysql replication

        2.1.3 limitations of database replication

    2.2 virtual machine replication

        2.2.1 remus

2.2.2 vmware replication

2.2.3 limitations of virtual machine limitations

3. Architecture

3.1 FALCON

3.2 KVM

3.3 QEMU-KVM

3.4 Summary

4. Implementation

4.1 Intercept the network packet

4.2 Process the network packet

4.3 Support concurrent connection

5. Evaluation

5.1 Database result

5.2 Virtual machine result

6. Conclusion and Future Work

Chapter 1

Introduction

In recent years, applied researchers have become increasingly interested in distributed systems. Distributed system is a collection of autonomous computers linked by a network, it uses software to produce an integrated computing facility. It not only enhances the performance of computing, but also provides higher availability and improved reliability. As a matter of fact, more and more

corporations begin to build their own distributed systems in data centers to support tens of thousands transactions every day.

Replication in computing is an essential region for distributed systems over the past few years. To ensure the data or state would exist all the time even in the case of server crash, data replication and computation replication techniques are well researched for many years. To guarantee the data and state consistency in distributed systems, many deterministic replication protocols like Paxos have been introduced in research.

1.1 State Machine Replication and Paxos

State machine replication is a general method for implementing a fault-tolerant service. Through replicating the state of servers and coordinating client interactions with replicas, state machine replication could provide high availability servers for the applications. In this model, applications are treated as a deterministic state machine, applying same input requests in the same order, the application would execute these requests and reach the same exact state. The replication here means many nodes keep the same state in different physical machines. Of course, the state machine replication model has the ability to overcome the single point of failure issue. As a matter of fact, state machine replication is a technique to improve the availability, not its performance. In most cases, the single point server would outperform a replicated system since it needs some consistent operations.

Paxos is a typical and famous state machine replication protocol in research region. It is also a milestone for the distributed systems. However, the original version Paxos is difficult for readers to understand as it takes ten years for researchers to implement it. In fact, this protocol is the simplest and most obvious in distributed algorithms. In this section, I will briefly explain the thoughts of this protocol.

There are three roles in this consensus algorithms, proposers, acceptors and learners. They have different responsibilities and privilege levels. And a single process often acts as more than one role, they could be a proposer in period of time and become the acceptor in the next timestamp. A proposer has the right to choose a value and regard this value as a proposal. Then it sends this proposal to all the acceptors. The acceptor must accept the first proposal it receives and respond with an acknowledgement to the proposer. The proposer will wait for the acknowledgements from the acceptors, once the acknowledgements reach the Quorum, this proposal becomes effective. Learners have no privilege to propose or agree a proposal. They act as a replication factor for the protocol. When a proposal has been agreed by the acceptors, the learners should take immediate action to execute it. As a result, they could improve the availability of the whole system. However, as all the proposers could choose a value and propose it, it always leads to conflicts between them. So most Paxos protocols introduces leader to make progress, in this situation, only the leader could propose the value and forward this value to the acceptors.

1.2 Limitations of traditional Paxos

Unfortunately, the high consensus latency of Paxos always makes system suffer. The big latency is caused by the network traffic. Network packet goes through TCP/IP layers so this process is taking place with the involvement of OS kernel. However, in a round-trip messaging, the processor context switches are very time consuming. When a message arrives at a compute node, a processor must be interrupted and software must discover the application that must process the new message via protocol stacks. Once the discovery takes place, the application must be context switched and the data is copied into the applications buffer before the message can be processed. In other words, involving the kernel will require the kernel threads to be scheduled to handle the TCP/IP, which is out of control of the protocol in the application layer.

The above-mentioned TCP/IP latency could be acceptable for leader election or other operations not sensitive to the time. But for the database or the real-time software, like the oracle database running in the bank, the latency time would be the big bottleneck for these systems. No doubt, no one is willing to add such a period of time to make the system very slow. But the demand for consensus protocol is so urgent that many corporations have no choice but to tolerant it.

Furthermore, with the increase of the replica group size, the latency will grow linearly. Because there would be more messages needed to send to the backup servers. The kernel threads should spend lots of time processing these packets. Of course, the context switches become more frequent.

1.3 Remote Direct Memory Access

The TCP/IP latency becomes the most serious issue for the Paxos protocol. To ease the influence of the TCP/IP network latency, we introduce the Remote Direct Memory Access (RDMA) device in dissertation. It is a direct memory access technique from the memory of one computer into that of another without involving either one's operating system. This permits high throughput, low latency networking, which is useful in distributed systems.

RDMA device is a network adapter connected directly with the network twine. It has the functions to wrap the messages and unpack the packets. In other words, the Network Interface Card in RDMA server is existing independently. It possesses some abilities like the Operating System. Unlike the OS needs to switch the context to get the network message, it eliminates the need to copy data between application memory and the data buffers in the operating system. RDMA is a developing technique widely used in companies' data centers. The RDMA message has the information of data link layer header, transport layer header and logical layer header in its packet. Although the structure of packet between RDMA and TCP/IP is very different, RDMA hardware device could support RDMA protocol like Infiniband and RoCE, and the traditional TCP/IP protocol simultaneously.

The work flow of the RDMA could be illustrated as follows: When an application in one host wants to execute the write or read operation, there is no need to copy the data from the application layer to kernel. RDMA request would send the message from the application space to Network Interface Card. The NIC reads the information form the buffer of itself, and transports it to another remote NIC. The

information transported in the network includes destination memory address, the key to access the memory and the data itself. The destination NIC gets the message and ensures the message has the correct key to access the memory. Then it would copy the data to the application's buffer. The work flow clearly shows zero-copy in operating system and few CPU involvements. According to the data collected in experiment, the one round-trip is 2~3 us in RDMA protocol compared with 50 us in TCP/IP.

1.4 Purpose of dissertation

Due to the high latency of the traditional Paxos protocol, it is very urgent to develop a RDMA-based Paxos protocol to solve the problem. This protocol should be reliable and secure, and it could support in various situations. For example, it could support the reliable replication for databases and virtual machines.

Database replication is a very hot topic in industry. Databases always store the important information for corporations. All IT companies and banks will use databases to fully make use of resources. Using database in corporations have the following advantages: sharing the data, reducing the redundancy of the data, and making the maintenance easier. However, database failure is a very headache problem for many years. To achieve the high availability of database, we should develop a reliable and synchronous replication protocol for it.

Virtual machine is another essential region requiring replication. Server virtualization has emerged as a powerful technique for consolidating servers in

data centers. It not only provides efficient and convenient resources containers, but also can be migrated smoothly between physical machines. For these advantages, it is an inevitable trend for enterprises to use this technique in data centers. At the same time, the service availability issue becomes a major concern in consolidated server systems using virtualization. The reliable replication protocol is also very urgent for virtual machine.

1.5 Organization of the dissertation

The dissertation addresses the single-point of failure problem in databases and virtual machine with RDMA-based Paxos protocol. Its organization is as follow.

Chapter 2 discusses some related work to achieve high availability of databases and virtual machine, and presents the limitations of these methods.

Chapter 3 introduces the design of the whole replication system, the architecture of database and virtual machine replication.

Chapter 4 describes the implementation details of the replication protocol and the method to integrate the protocol into database and virtual machine.

Chapter 5 reports the experiment results of my dissertation. It includes the analytical model, and the performance comparison between normal case and the replication method.

Chapter 6 draws the conclusions of my dissertation and suggests some future works on virtual machine replication.

Chapter 2

Review of literature

Replication in databases is very popular in markets. Some mainstream databases like Redis, Mysql and Mongodb, all have their own replication mechanism. Often, the default replication method is very simple, modifying some configuration files is enough to serve as a replication system. In the next section, I will briefly introduce the replication mechanism for the RDBMS database Mysql and key-value database Redis. Both of them are open sourced and developed for many years. But the replication methods are different.

Virtual machine replication seems much harder compared with the database replication in commercial. High availability systems that aims to replicate the

virtual machine could be realized by using special hardware or some customized software. Even the hardware method or the software method requires expensive and complex redundant servers. The system could transparently survive failure and provide fault-tolerant servers. For Xen, Remus has been widely used to guarantee high availability using software-based scheme. And VMware is also developing its own replication method in its product vSphere. I will also describe these two techniques in the section two.

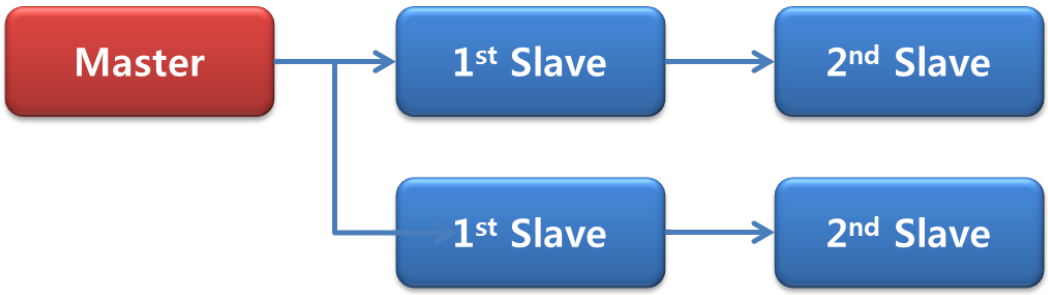
2.1 Database replication

2.1.1 Master-slave replication for Redis

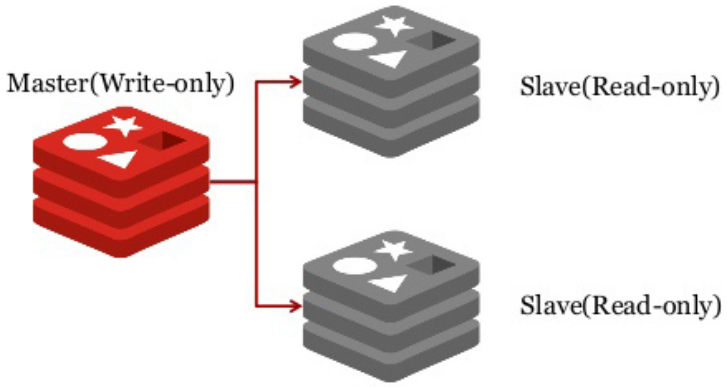
Redis is an open-sourced, in-memory, key-value store database original developed by VMware corporation. It is the most popular NoSQL database according to monthly rankings by DB-Engines. Due to its in-memory nature, the storage speed could be very fast and the get operation is efficient. This nature allows the Redis perform better compared with other databases that need to write to disk every operation. But the durable cannot be guaranteed in this process because the information in memory will lost with the crash of servers.

Redis supports master-slave replication, it allows the slaves to be exact copies of master servers. The replication mechanism is started with the version 2.8 of Redis. Like most of the database replication, it is an asynchronous replication, it means the master Redis server could handle the requests from the clients, at the same time, it would forward the requests to the slaves. In other words, the replication is

a non-blocking process in the master node. At the node of slave Redis, the process is also non-blocking, too. The slave node could handle the requests from the master, these requests are collected and stored in the dataset. Slaves are able to connect with each other to change the information, even the connection with the master is cut off, the slave could still receive information from other slaves as long as at least one slave is connected with the master Redis. Furthermore, it could no doubt reduce the pressure of master node as no need to forward the queries to every slaves. The architecture is just like the figure below. The slaves are chained together in a line.



Although the slave Redis could not handle write operations in its node, the read operation is permitted. It means the clients could connect with the slaves to get the correct information. The throughput of read operation is increasing linearly with the number of slave nodes.



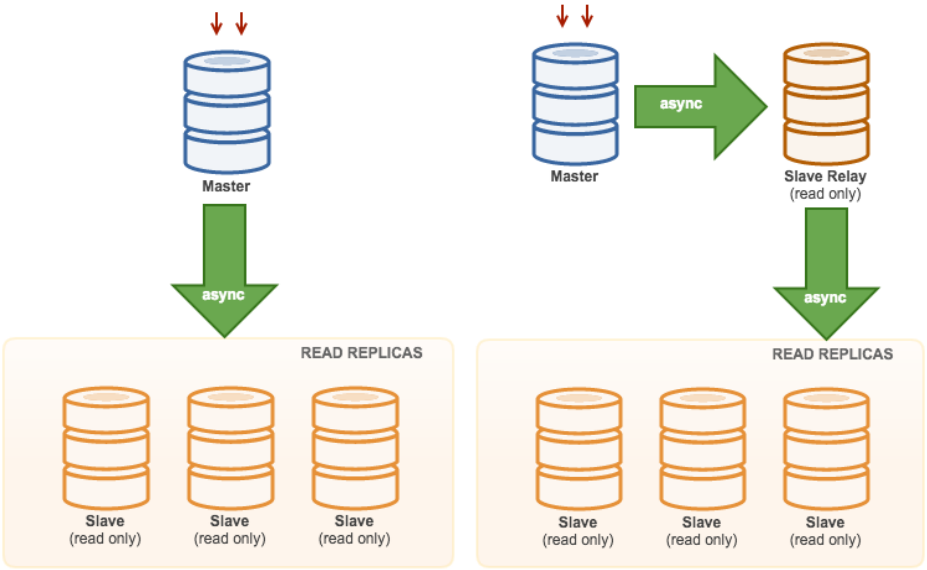
2.1.2 Replication for Mysql

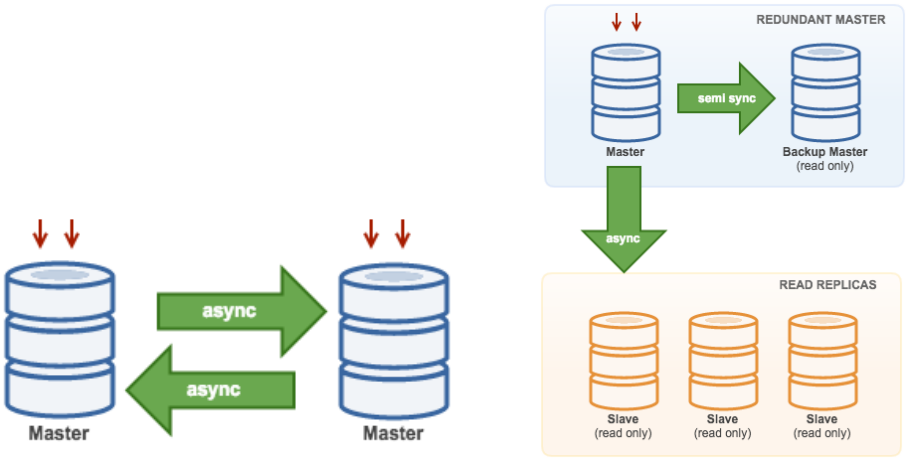
Mysql is an open-sourced relational database management system, unlike the key-value store database Redis, the data is stored persistently in the disk rather than memory. As a matter of fact, many applications use Mysql database as the storage tool while many corporations use it in large-scale websites, including famous companies Facebook, Twitter, YouTube and Google. Now the new release version Mysql supports two types of replication: master-salve replication and master-master replication. These two methods could be combined to produce many topologies.

Master-slave replication is similar with the Redis replication. Only one master could receive write queries from the clients. Others could only handle read requests. The slaves could be chained together to make the performance better.

Master-master replication is different from master-slave replication in the way that both Mysql server could act as master and handle write queries. But up to now, the Mysql server has no resolution to solve the conflict between two master

nodes. The programmer should guarantee no collisions himself to support this topology. In this way, the master-master replication mechanism seems a little immature. At the same time, another master-master replication method is proposed called master with backup master. In this situation, the backup master would not receive the write queries all the time, but the asynchronous replication is changed to semi-synchronous. Master node sends every update to backup master and waits for its acknowledgement. It is a blocking method. The steps are described as follows: The master sends update to backup master. Backup master receives update, flushes the change to the disk and sends acknowledgement to master. The master who receives the acknowledgement begins to commit transaction. Without doubt, this method has a performance impact, but the conflictions are disappeared and the risk of data inconsistency would not exist.





2.1.3 Limitations of database replication

In conclusion, Redis now supports master-slave replication while Mysql supports master-slave and master-master replication. These two methods both have some limitations.

For the master-slave replication both in Redis and Mysql database, the replication seems not very reliable because it is a kind of asynchronous replication. It cannot guarantee the state consistency between the master and slave. In some situations, the master may have applied some queries in its database while the slave has not received log information. It would lead to the data inconsistency between two nodes. If the master node crashes in this period of time, obviously, the slave node cannot take over and handle the following tasks.

As for the master-master replication for Mysql, the semi-synchronous replication is more powerful than the former one. Although multi-master cannot solve the confliction problem, master with backup masters is better and the state is consistent between all masters. However, this approach is time-consuming and the response time is too long. When lots of queries are arriving at the same time, the

masters cannot process them efficiently. And with the number of masters increasing, it would burden the operating system in master to send and receive more TCP/IP messages.

2.2 Virtual machine replication

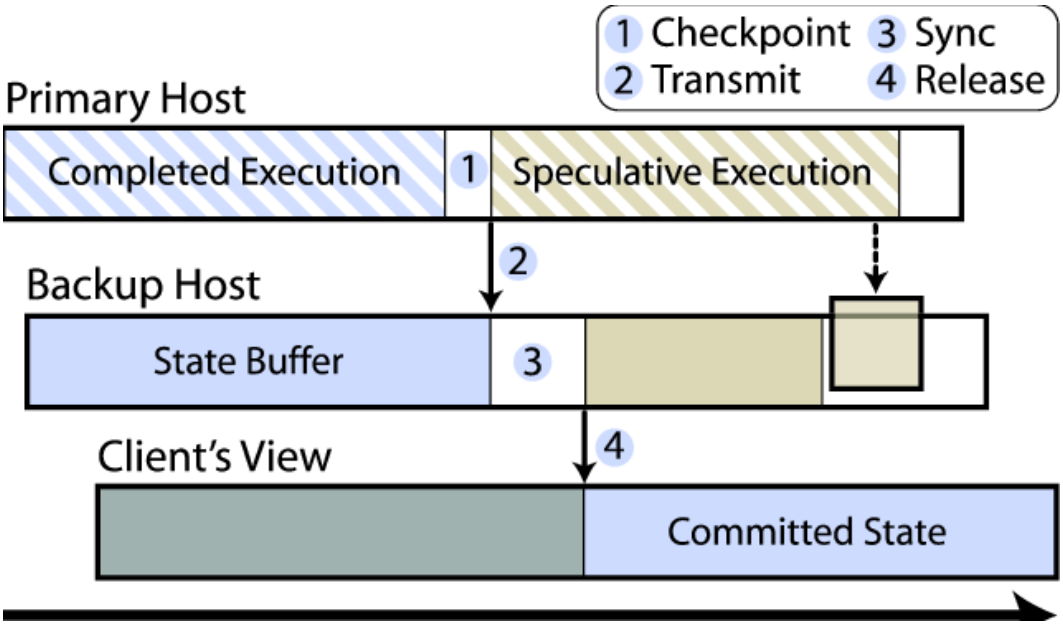
A common approach to implementing virtual machine replication is the primary and backup approach where the backup server could take over all the tasks when the primary server fails. The state of the primary server and backup are always identical at all times. In Xen, Remus has been used widely for many years. The main idea of this technique is the checkpoint of virtual machine. It needs to transport the changed state from primary to backup in a small period of time. A different method for replicating virtual machine is the state machine approach. It is supposed that two virtual machines are started at the same initial state and it will receive the same requests at the same order. For the deterministic state machines, this approach is viable. VMware corporation developed this approach in its product vSphere 4.0 and made fault tolerant virtual machine available. In this section, I will discuss these two approaches in detail.

2.2.1 Remus - replication for Xen

Remus is a software system that provides high availability for applications or operating systems on commodity hardware. This approach is taking advantage of the method in live migration for virtual machine. Live migration is a feature provided by the virtual machine to migrate a guest operating system from one host

to that of another. The frequency of migrating every snapshot is as fast as every 25ms. The work flow of this approach can be described as follows. Two virtual machines are located in two different physical machines in local area network. Only the primary server could receive the network packets from the clients. Suppose one client wants to create a connection with the backup server, the backup server would redirect this connection to the primary. In this way, the backup server would only receive the information from the primary. In other words, the state of the backup can only be changed by the primary. However, for the primary server, without directly responding to the clients, the virtual machine will buffer these output. After a period of time, around 25ms, the virtual machine would suspend itself and collect all the changed state, including memory, CPU and I/O device state. The changed state is caused by the execution of inputs from the clients. The CPU and I/O state can be shipped to backup server efficiently as there are few information in them. After that, the primary continues to accept requests from the clients. Another thread in primary virtual machine's host operating system is responsible for sending these changed state to the backup server. The backup server who receives these buffered state information begins to sync immediately. It would change the CPU and I/O device state to the new state, and change some dirty memory in this period. Finally, when all the above-steps done. The backup server would send a signal to the primary to release the buffered output to the clients. As a result, the primary and backup virtual machine could hide the internal issues to the outside world, if the primary fails in this process, to the sight of the outside world, the state is consistent as the buffered

output are not released to the clients. The clients would think the previous operations are not well performed in virtual machine and resend it to virtual machine again.



2.2.2 Limitations for Remus

The primary and backup approach in Xen could effectively replicate the virtual machine and make fault tolerance come true. However, there are still some limitations for this approach.

Most importantly, it needs to consume large amount of resources to ship changes of all state of primary to the backup nearly continuously, especially the changed memory. For the memory, even modify one byte, the primary must ship the whole memory page to the backup (the memory page size is often 4KB in Linux). No doubt, large changed memory would burden the host machine with state

transmission. If the bandwidth between two virtual machines are not large enough, the frequency of migration should be decreased.

Furthermore, suspend the virtual machine too frequently will reduce the service time of the primary, and in the suspension period, the resources cannot be allocated to other applications, it means the CPU usage would be very small. It will lead to the degradation of performance.

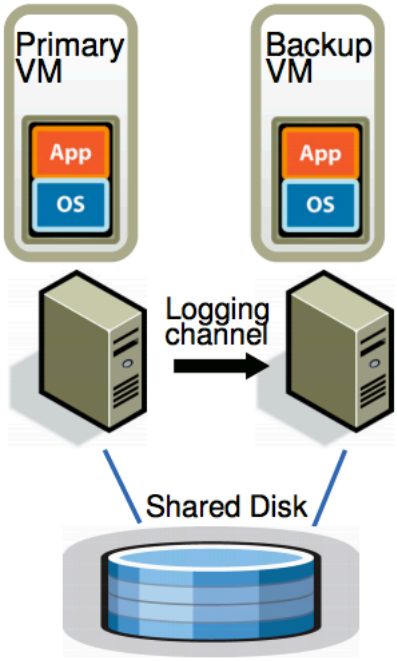
Thirdly, it would suffer from inconsistent primary. If the old primary fails and comes back. Then the backup becomes new primary. In this situation, it is difficult to determine which one is more up-to-date. In other word, it is difficult to determine an outdated primary. It would lead to inconsistency between primary and backup. And for these two virtual machines, they both regard themselves as the primary and begin to process the requests.

2.2.3 VMware vSphere replication

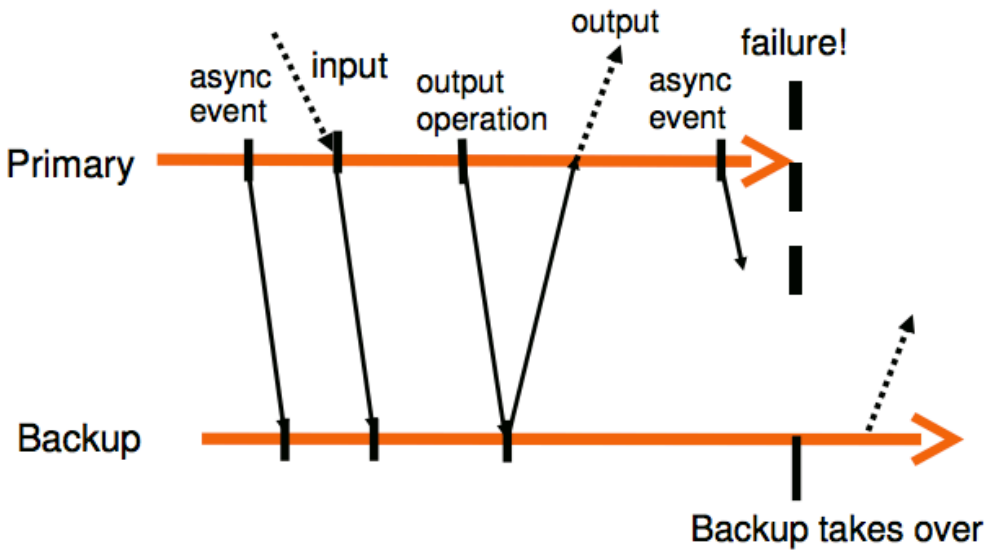
VMware corporation imbed the replication into virtual machine with the deterministic log replay approach. Like the Remus, for the virtual machine which we desire to replicate, we run another backup virtual machine on a different physical machine. The backup server would execute the same inputs with the primary server with a small time lag. Of course, only the primary server would send and receive network packets. The backup server remains silent all the time and only communicates with the primary server. The communication here means the logging channel in vSphere. To guarantee the backup server executes identically to the primary, we should ensure all the operations must be

deterministic. For those non-deterministic operations, we must ensure those can be executed in the same way.

The input and output is a very essential part for the log replay replication, unlike the checkpoint method of Remus, both primary and backup produce the output in the process of execution. It needs to write the results to the memory and disk. As for the input, the primary receives input from clients while the backup receives from the primary. In order to simplify the output part, VMware utilizes the shared storage to store the disk. Both the primary and backup could read the information in shared disk but only the primary has the privilege to write to it. The primary's output would first be buffered until the backup virtual machine has received and acknowledged the log entry associated with the operation producing the output. The backup's output which should be sent to the clients will be dropped as the primary has already sent to the client side. The architecture of the fault-tolerant virtual machine and the protocol are displayed below.



Apart from the deterministic events in the operating system, there are still many non-deterministic operations such as the operations to read the clock information. How to correctly capture all the non-determinism to ensure the execution deterministic is the biggest challenge in this technique. The vSphere provides a log file to record all the non-determinism. When the input is captured by the vSphere, it would judge immediately if this operation is deterministic. If not, it means the primary needs to provide sufficient information to allow the backup server could reproduce this state. During replay, the event is delivered at the same point in the instruction stream. VMware deterministic replay implements an efficient event recording and event delivery mechanism that employs various techniques, including the use of hardware performance counters developed in conjunction with AMD and Intel.



2.2.4 Limitations of VMware log replay

It is obvious that compared with the Remus, the VMware's log replay approach is more reliable in network layer as it only needs to send small amount of log information. With this log file, the state between two virtual machines are identical with each other. There is no need to ship all the changed memory which consuming large amount of network bandwidth resources.

However, there are still some limitations for this approach, first of all, as I mention above, the log replay approach cannot effectively resolve the problem of non-deterministic operations. For the non-deterministic operations like clock functions, the primary server must collect enough data to reproduce the same state in backup server. In this way, the developer must know all the non-deterministic operations first. Otherwise, the primary server may lose sight of some non-deterministic operations which contribute to the inconsistent between two machines.

Secondly, it cannot support multi-thread in guest operating systems. For the multi-thread applications, it would produce some concurrency bugs if some codes are not locked when executing. In the virtual machine, multi-thread would account for the inconsistency between virtual machines. The developer cannot insure the operations in multi-thread are executed in the same order. So the results of execution may vary.

Thirdly, this approach also has the split-brain problem like Remus. It cannot effectively determine which virtual machine is more up-to-date when the old primary fails and comes back while the backup server becomes the new primary. Although it describes a good method to determine who is the primary by using the shared disk. The shared disk is just like the third party to judge who is the primary like the Paxos protocol. However, if the log replay approach is running in two different virtual machines who are not willing to share the disk, the third party decider becomes hard to find.

2.5 Summary

In this chapter, we studied the related work on the database and virtual machine replication. The limitations of database replication lie in the high performance overhead and low consistency. The primary and backup approach is simple but unreliable while the semi-synchronous replication would add much latency in one operation. The virtual machine replication is mature and complex, Remus needs to transport large amount of memory and CPU state which heavily burdens the host operating system. The VMware log replay cannot support multi threads and high

availability cannot be guaranteed with only two machines. Our proposed replication service should be designed to overcome these limitations.

Chapter 3

Architecture

This section introduces the architecture of two key techniques in this dissertation, Kernel-based Virtual Machine, with reference to its network I/O architecture in QEMU-KVM and the FALCON SMR system.

3.1 FALCON

FOLCON is a typical Paxos protocol implemented with the RDMA library libibverbs. We run replicas in our cluster where all servers are connected with a switch. Every replica registers a memory for itself and connects with each other using RDMA QPs. So, every replica has the key to access the memory in other nodes.

Once the FALCON starts in one node, it would try to establish reliable connections with other replicas. When all the connections are created between replicas, it would elect a leader node which proposes the order of requests to execute. Other replicas are acting as acceptors in our algorithm. No learners exist in system. An arbitrary number of clients in LAN or WAN send network requests to the leader and get responses.

FALCON is a strong consensus algorithm consisting of four main steps. On receiving a client network request, the leader node first intercepts the requests and get the real data from the relevant packets. The second step is local preparation, add the viewstamp information for this consensus log and write it to a local parallel logging storage on SSD. In third step, it would invoke a RDMA-based distributed consensus protocol on this request to enforce that all the replicas see the same input. The fourth step is the leader collecting the ACKs from the replicas, once it reaches Quorum, it means the consensus is completed. Benefiting from RDMA and FALCON's fast protocol, each consensus process can take less than 10us. Compared with TCP/IP-based Paxos protocol, whose latency was always over 600us, FALCON has the advantage of low performance overhead.

3.2 KVM

We use Kernel-based Virtual Machine as the platform of our study mainly because of the full capacity of user-space tools in its I/O device emulator process. KVM is a full virtualization or hardware-assisted virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of two modules, namely, kvm.ko and an architecture dependent kvm-

amd.ko or kvm-intel.ko module. Under KVM, every virtual machine is a regular Linux process handled by the standard Linux scheduler by adding a guest mode execution which has its own kernel and user modes.

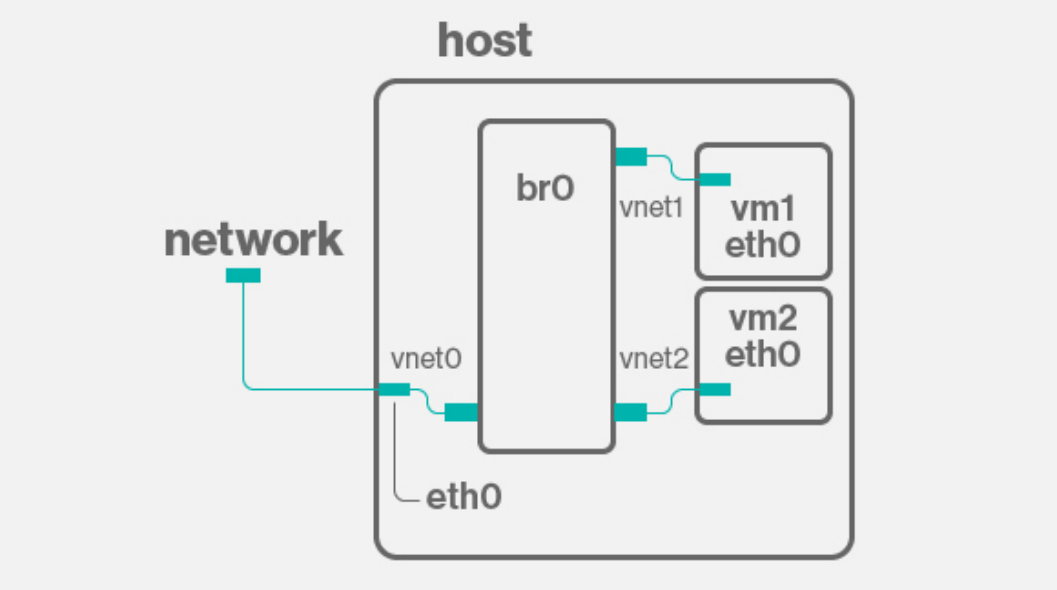
KVM module has been merged into the Linux kernel after the version 2.6.20. Full virtualization means there is no need to modify the operating systems like the Paravirtualization Xen. Furthermore, a wide variety of guest operating systems could be run in the KVM including Linux, OpenBSD, Windows and FreeBSD benefiting from the full virtualization technique. Through comparing KVM and other products such as VMware and Xen, it is easy for us to find KVM could support in more situations and has better performance behavior. More accurate, in some performance evaluations, it is observed KVM has the same performance level as the leading proprietary technologies.

KVM is a kind of hardware-assisted virtualization, form its name, we could know only the specific hardware could support this feature. In default, this feature is disabled in the physical machine. Enable this function needs to reboot the reset the machine as a result. Of course, in the operating system, the relative module should be loaded into the kernel.

3.3 QEMU-KVM

KVM is responsible for the CPU and memory virtualization in the physical machine. However, a virtual machine is not a process only possesses the CPU or memory resources. As a matter of fact, KVM also requires a modified QEMU which is located in the user space to provide emulated I/O devices. We call this application QEMU-KVM.

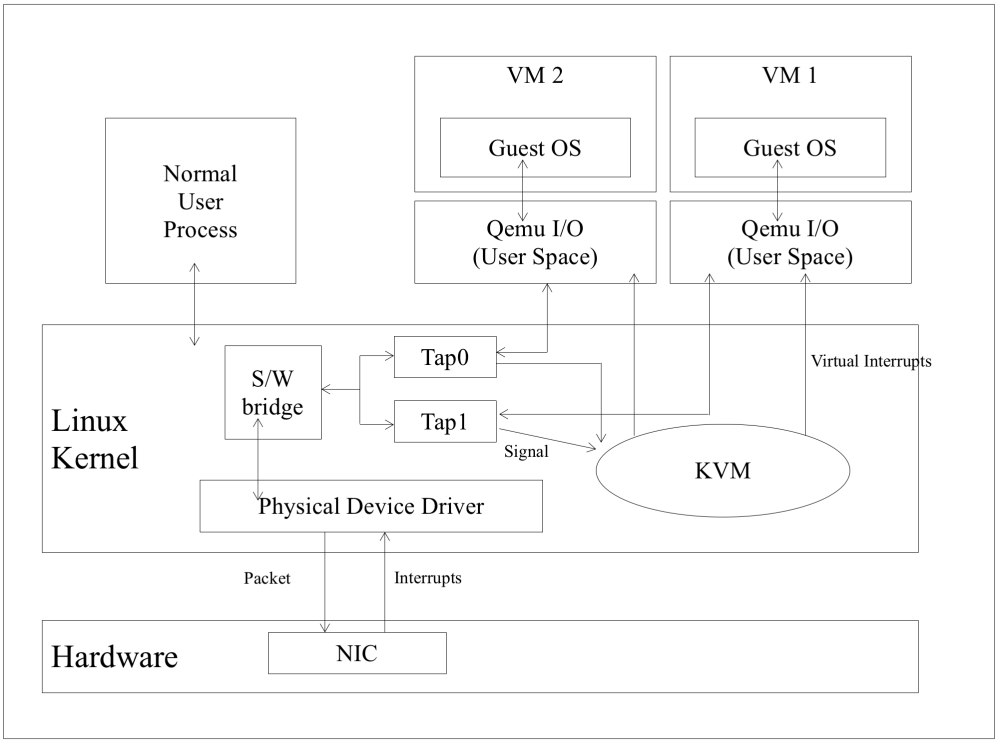
In our design, our clients are located in the local area network in other physical machines. In other words, the applications running inside the virtual machine could be access by the clients outside the host machine. To make this come true, we install a virtual Ethernet bridge in our host machines. Through binding the host machine's network interface with the virtual bridge, binding bridge with virtual machine's network interface, and allocating a static IP address for the virtual machine, the clients in other hosts could access this virtual machine easily. What is more, when binding an interface with the bridge, the bridge would also require a virtual front end interface to connect with the back end interface in the virtual machine. The architecture of the virtual Ethernet bridge is shown below.



The networking in KVM is implemented in the user space QEMU-KVM application. A typical packet flows through the KVM virtualized host in the following way. When a packet arrives at the physical NIC in the host, interrupts generated by NIC are handled by the physical driver. Then the physical NIC device driver handles the interrupts by transferring the packet to the virtual

Ethernet bridge, which forwards the packet to the appropriate virtual machine's front end interface.

The front end interface in QEMU-KVM we used is the TAP device which are entirely supported in software. It simulates an Ethernet link layer device which operates the Ethernet packets. The TAP device would create a file descriptor in host machine when we start it. And this file descriptor is responsible for receiving all the packets from the local area network. When the network packet is arriving at the TAP device, the TAP device gives a signal to the KVM driver which sends a virtual interrupt to the QEMU in the guest notifying it of the new packet. Once receiving the virtual interrupt, QEMU-KVM delivers the packet to the guest operating system's network stack. The architecture of the whole network flow is shown below.



3.4 Summary

In this chapter, we introduce the architecture of this dissertation. FALCON is a RDMA-based Paxos protocol responsible for the input log replication. KVM is the platform for us to achieve virtual machine replication while the QEMU-KVM is used to emulating I/O devices in the virtual machine.

Chapter 4

Implementation

In this chapter, we describe the implementation details of the virtual machine replication. We first discuss the process of getting the network packet from the QEMU-KVM. Then, we turn our attention to the steps to get our needed packet the its real data part. Finally, the replication process achieved in FALCON.

4.1 Intercept the network packet

As we mentioned above, we use the TAP virtual network device as the front end interface to receive the network packet from the virtual bridge. So, how can we

get the network packet from this virtual network device becomes the most urgent question.

The tap_send() function in QEMU-KVM is called to check whether there is any frame received by the TAP device. And the tap_receive() function is used for receiving frame from the virtual machine. These two functions are jointly concerned with a file descriptor. The details of interception process could be described as follows. The file descriptor is created before the activation of virtual machine. It is responsible for receiving the packet outside the virtual machine. The system call read() would be invoked and return a non-zero value when there is any packet received by this file descriptor. The information of the packet would be written into the memory and read() function would return the length of the packet. Then, the tap_send() function would judge if the length of packet is not zero. If so, it would forward this packet to the virtual machine asynchronously. With adding some code in the tap_send() function, it is easy for us to intercept all the network packet sending from the clients.

4.2 Preprocessing of the packet

As the TAP device is an Ethernet device running in the data link layer, the packet we intercepted contains the information of the Ethernet header, IP header and TCP or UDP header. The real data part is followed after these headers.