硕士学位论文

云计算系统故障注入平台的研究与设计

RESEARCH AND DESIGN OF FAULT INJECTION PLATFORM FOR CLOUD COMPUTING SYSTEM

柴森

哈尔滨工业大学

2016 年 6 月

国内图书分类号：TP302.8　　学校代码：10213
国际图书分类号：681.39　　密级：公开

工程硕士学位论文

云计算系统故障注入平台的研究与设计

硕士研究生：柴森
导　　师：左德承教授
申请学位：工程硕士
学　　科：计算机技术
所在单位：计算机科学与技术学院
答辩日期：2016 年 6 月
授予学位单位：哈尔滨工业大学

Classified Index: TP302.8
U.D.C: 681.39

Dissertation for the Master Degree in Engineering

RESEARCH AND DESIGN OF FAULT INJECTION PLATFORM FOR CLOUD COMPUTING SYSTEM

Candidate: Chai Sen
Supervisor: Prof. Zuo Decheng
Academic Degree Applied for: Master of Engineering
Speciality: Computer Technology
Affiliation: School of Computer Science and Technology
Date of Defence: June, 2016
Degree-Conferring-Institution: Harbin Institute of Technology

摘　要

随着云计算技术的发展，各种各样的云计算平台都涌现了出来，各种研究机构和大公司都推出了自己的云平台,给用户提供方便的可伸缩的云计算服务。云计算平台是云计算中的核心基础设施，所有的云服务与应用都需要部署在云平台上，云平台可以管理底层的硬件资源例如网络资源、存储资源和计算资源。虚拟化层又可以把一个个物理硬件虚拟成多个设备提供给上层的软件使用。云计算包含多个层次，分为 IaaS 层、PaaS 层以及 SaaS 层。但是，随着云计算系统的复杂度和规模的增加，云系统在运行过程中可能出现的故障和错误也越来越多，严重的故障会给公司和用户都带来严重的损失。

本文针对云计算系统的健壮性评测这一问题，构建了针对云计算系统的故障注入平台，采用分布式的架构，根据典型的云系统的分层架构，把故障注入分为多个层次，分别是分布式计算层，虚拟化层，云平台管理栈层，本文对这三层设计了统一的故障注入框架，并将故障注入的过程标准化，给测试人员提供多种接口方便测试的进行，一种是图形化的实时注入接口，一种是脚本方式的批量注入。并且可以对系统的简单故障进行检测，并能收集故障注入的信息。

之后，本文针对定义的三层的故障层次，对分布式计算层，虚拟化层，云平台管理栈层选取了典型的实际系统，分层研究了各个层次的结构特点和容错机制，然后设计了针对这种特点的故障模型和具体的故障注入工具。

最后本文在自己搭建的由 CloudStack，Xenserver，KVM，Hadoop，Spark 组成云系统上进行了故障注入的测试实验。先分别对各个层次进行了故障注入的实验，然后对不同层次的故障进行了跨层次的故障注入实验。通过实验，我们不仅验证了一些系统自动的容错机制，还对系统的一些不足提出了改进的意见。

关键词：云计算；分布式计算；虚拟化；故障注入；容错性测试；

Abstract

With the development of cloud computing technology, a wide range of cloud computing platforms have emerged out, a various of research institutions and large companies have launched their own cloud platforms, to provide users with convenient and scalable cloud computing services. Cloud computing platform is the core of the cloud infrastructure, cloud services in all applications need to be deployed on cloud platforms, cloud platform can manage the underlying hardware resources such as network resources, storage resources and computing resources. The virtualization layer and can put a physical hardware into multiple virtual devices to the top of the software. Cloud computing includes muti-layers, IaaS, PaaS and SaaS layer. However, with the increasing complexity and scale of the cloud systems, cloud system failures and errors that may occur during operation, more and more serious failures will give companies and users serious losses.

This paper reviews the robustness of the cloud computing system this problem, build a cloud computing system for fault injection platform with a distributed architecture, based on the typical hierarchical structure of cloud systems, the fault injection into multiple levels, respectively distributed computing layer, the virtualization layer and the cloud management stacks layer. This paper designs a unified fault injection framework, standardized the fault injection process, giving testers a variety of interfaces to facilitate testing conducted a species is a graphical interface to the real-time injection, one is scripted batch injection. And simple fault can be detected by our system and we can collect informations on fault injection.

After that, this paper selects typical practical systems from three levels of fault models, distributed computing layer, the virtualization layer and the cloud management platform stack layers, then we investigated their features and fault tolerance mechanisms, and designs tools for the injection fault model and the specific characteristics of this fault.

Finally, we use CloudStack, XenServer, KVM, Hadoop, Spark composition to build a cloud system fault injection testing laboratory. First of all, we conducted fault injection experiment for each levels, and then the different levels of cross-level fault injection experiments. Through experiments, we not only verify some system of automatic fault tolerance, but also on some of the deficiencies of the system and put forward suggestions for improvement.

Keywords: cloud computing, distributed computing，virtualization，fault injection，fault tolerance testing

目　录

第1章 绪　论

1.1 课题背景

1.1.1 课题来源

本课题来源于国家 863 计划重大项目“云计算测试与评估系统研制”。

云计算已经成为了大型 IT 公司提供的典型服务模式，例如 IBM，Microsoft，Google，Amazon 和阿里云，他们都是云计算服务的提供者。目前，各行各业对于计算能力的动态的需求呈增长的态势，云计算这个行业的市场规模也越来越庞大。对于消费者来说，云计算是一种潜在的代替传统自己管理数据中心和机房模式的解决方案。它可以减少用户的使用和管理成本，而背后维护庞大的计算资源的任务都落在了背后的 IT 公司的身上。他们可以按照消费者的需求动态的提供服务，之后计算出费用。消费者所做的就是使用这种便捷的计算资源。

但是随着云计算平台越来越庞大，它也变得越来越复杂，它的可靠性和可用性成为了云计算服务提供商的一个很大挑战。例如亚马逊公司在 2011 年 4 月 21 日发生了超过两天的 EC2 大规模停机事件[1]，该事件是由于 Amazon 的故障处理机制的问题导了大规模停机，该事件影响了众多国外知名网站的访问。如果我们能通过注入网络故障或者主机故障的方法来模拟这种情形，也许可以提前发现容错机制中的漏洞，避免在系统正常运行时发生意料之外的故障。在构建云平台之前或者构建同时，如果我们能够对云平台的健壮性做到很好的测试，这就可以防止它在提供服务时出现巨大的问题。然而由于云平台的复杂性，与种类的繁多，对于测试人员缺乏一个统一的方便的测试框架来对云计算平台的健壮性和可用性进行评测。因此，我们需要研制一个这样的平台来对云计算系统进行容错性、健壮性测试，本课题就是在这种背景下提出，我们要构建一个统一的云计算故障注入平台来方便的评测一个完整的云计算系统。本文在分析了现有的云平台系统的各层之后，搭建了较为典型的云平台的环境，然后设计了统一的故障注入框架，并针对云平台的各个层次，分别提出了针对各个层次的故障注入工具，抽象出了针对各个层次的故障模型。这些工具可以针对各个层次的特点注入各自的故障，模拟真实运行环境中可能出现的各种错误场景，来达到评测云计算系统的可靠性和可用性的目的。

1.1.2 课题目的与意义

在现实场景中如果云平台发生了较大规模的故障，它造成的损失会非常巨大，因为有越来越多的服务都部署在了云平台上，云平台的故障可能会给用户造成难以估计的损失。国内外有很多人都投入到了对云计算的相关的研究上来，但是对于云计算的可靠性的评测方面还不是很多。在国外有一些人在这方面做了一些工作。

目前随着云计算技术的发展，各种各样的云计算平台被提出，虽然种类有很多种，但是他们的基本架构是类似的，因此，我们可以从这个角度出发研发一个针对云计算系统的较为通用的故障注入系统，设计出较为通用的故障集和故障注入方法，能够对云计算平台各个层次上面的各个组件进行故障注入和测试，通过故障注入这种手段，我们可以很好的评测一个云计算平台的可靠性，为云计算系统今后的设计和使用提供指导性的意见，设计出更为完善的容错机制，能在出现异常情况时做到有效的处理，提高云计算系统的容错性。

1.2 国内外研究现状

1.2.1 云计算平台简介

云计算是继 20 世纪 80 年代由大型机向客户端/服务器模式转变后，信息技术领域的一次革命性变化[2]。云计算平台是云计算中的核心基础设施，所有的云服务与应用都需要部署在云平台上，云平台可以管理底层的硬件资源例如网络资源，存储资源，计算资源。虚拟化层又可以把一个物理硬件虚拟成多个设备提供给上层的软件使用[3]，这可以提高资源的使用率，并且可以便于管理。云计算包含多个层次，分为 IaaS 层，PaaS 层以及 SaaS 层[4]。本文所说的云平台侧重于 IaaS 层和 PaaS 层。目前 Iaas 层常见的云平台有 CloudStack[5]，Openstack[6]，Eucalyptus[7]等。虚拟化层有 Xen[8]，KVM[9]等。分布式计算平台有 Hadoop[10]，Spark[11]等。

在商业层面，很多公司都拥有自己内部或者外部的云计算平台，来给自己内部或者外部用户提供便捷的云计算服务。Google 的云计算平台包括 google 分布式文件系统 GFS[12]，MapReduce[13]编程框架，分布式锁机制 Chubby[14]以及针对海量数据的分布式数据库 BigTable[15]，这四种服务都拥有各自有效的容错策略，可以在分布式的计算环境中容忍不是很致命的故障。这 4 种系统相互协作给 google 内部使用者提供便捷的健壮的存储以及计算服务。Microsoft Azure[16]是微软基于云计算的操作系统，Azure 提供了一个灵活的云平台，可以

满足任何应用程序的需求。按需拓展，可以使用各种各样的数据库来存储不同特点的数据，例如可以使用关系 SQL 数据库，NoSQL[17]数据库，可以使用 Azure 健壮的消息传递功能来实现可扩展的分布式应用程序。

1.2.2 基于软件的故障注入技术研究现状

故障注入技术已经被广泛的用来对系统的可靠性或者容错机制来进行评估[18]。故障注入技术在对系统的容错机制进行测试方面有着巨大的优点，它可以人为的模拟平常很少出现的错误，使系统出现故障处理的场景，通过跟踪系统的处理流程和反应，我们就可以较为精确的测试系统的容错性。故障注入技术有很多种，有基于软件的，有基于硬件的，有软硬件结合的。因为我们后面主要使用基于软件的故障注入技术，因此我们着重介绍基于软件的故障注入技术（SWIFI[19]）的研究现状。

基于软件实现的故障注入技术使用特殊的软件方法来更改系统中的软件和硬件的状态来实现系统表现为硬件或者软件故障。较早的方法是 FIAT[20],它可以写脏任务的内存映像。选择的位置是用户在应用层选定的，地址是内存镜像中的物理地址，它可以通过编译器和装载器获得这些信息，但是它不能注入瞬时的故障。

一种叫 DOCTOR[21]的工具可以在实时分布式系统上注入处理器、内存、通信故障。处理器故障通过更改应用的可执行映像来实现，尤其是可以通过更改由编译器编译好的指令或者增加指令来实现。

在 FERRARI[22]的方法中，他使用了 unix 的 ptrace 函数来实现在运行时更改进程的内存映像并且可以在特殊的指令的地址上插入软指令。这个工具允许注入瞬时的故障，并且还分析了在 Sparc 工作站上的实验结果。

一种叫 FINE[23]的工具可以用来注入故障并且可以控制跟踪注入的流程，然而，这个工具需要目标程序的源代码，它更改了程序的可执行镜像来模拟内存故障，在特定的段的内存位置插入软中断。

目前来看基于软件方式的故障注入工具已经有很多，但大多都是针对操作系统级的故障注入，面向云平台的还很少。

1.2.3 云计算系统故障注入国内外研究现状

云计算中的故障注入技术是来评估云计算系统的可靠性和容错性的一种常用方法，很多研究者对云系统展开了故障注入方法的研究。他们大多使用的还是基于软件的故障注入方法来实现故障注入。

Le [24]等在虚拟化系统上使用基于软件的故障注入方法（SWFI），他们对虚拟机和虚拟机管理器注入了故障，并且在非虚拟化，半虚拟化，全虚拟化的不同平台上做了对比，他们对虚拟机化系统故障注入方面的挑战做了总结。

Ju [25]等构建了一种故障注入的原型框架来对 OpenStack 上的组件间的还有第三方软件间服务的通信来进行故障注入。他们在详细的分析了 OpenStack 的架构后，针对各个服务的特点设计了对应的故障注入方案。

Benz 和 Bohert [26] 展示了一种可靠性的模型来测试云的高可用性。他们使用故障注入的方法来使云组件崩溃，记录他们的崩溃的时间，对预期产生的影响进行建模。在不同的复杂场景的应用中，他们使用了不同的函数。但是，函数和组件间的关系被视为先验的知识，不是由故障注入的方法得到的。

Souza[27]等实现了一种针对 Eucalyptus 云平台的故障注入工具。他们对 Eucalyptus 的硬件和软件组件注入了在运行时和支持的某些恢复场景的故障，并且对对可靠性进行了分析。这套工具的缺点是只能对进程和硬件资源进行故障注入，而不能对云服务进行功能性故障注入。

Chaos Monkey[28]是运行在亚马逊 Web 服务（AWS[29]）上的一个服务，它可以找出自动扩展组（ASGS）并且可以停止每个组内的虚拟机实例。这个软件设计的很灵活，可以在别的云平台上或者实例组上使用。这个服务在上午 9 点到下午 3 点运行。虽然在亚马逊云上部署的应用在设计时已经考虑到了在某个实例离线后仍然会继续运行，但是在某些特殊的情形下，这些应用还是发生异常，为了发现这些异常，他们开发了 Chaos Monkey 这套系统。在 Chaos Monkey 的帮助下，他们检测到了很多的失败场景，对于一些异常的场景，AWS 会把它隔离和解决使得这种失败的场景不再发生。Chaos Monkey 随机的选择一个实例然后终止它，亚马逊云的 ASG 模块可以自动的检测到这个实例的终结然后自动的启动一个新的实例，并且和以前的实例使用相同的配置。Chaos Monkey 采用了 REST 风格的接口来给用户提供操作。

Luigi De Simone[30]对云计算生态系统中的故障的传播进行了分析。这篇文章讨论了目前在云计算生态系统的可靠性分析中已有的方法和一些挑战。然后分析了一些需要的技术和方法来预防故障在云计算生态系统中的传播。我们对整个云计算系统进行故障注入的思想与此类似。

Haryadi[31]设计了针对云恢复的一种新的测试框架 FATE(故障测试服务)和 DESTINI(说明性的测试定义)。使用 FATE，可以全面地测试面对各种故障时系统的恢复能力。使用 DESTINI，可以清晰简明准确的定义正确的恢复这种行为。它们使用这个框架对多种云系统进行了整合和测试，包括 HDFS 等系统。

国内针对云平台的故障注入的研究较少。

冯刚[32]设计了针对 Xen 虚拟机的多层次故障注入工具，可以针对 Xen 的超级调用，事件通道，内存故障注入故障，还提出了针对虚拟机内核，虚拟机寄存器，Dom0 的管理功能的故障注入工具。较为完整的覆盖了 Xen 虚拟机的故障集。

麻彦东[33]设计了一个针对 Xen 虚拟机的故障注入平台可以对 Xen 虚拟机的寄存器，内存，和超级调用注入故障，但没有考虑对 KVM 等虚拟机管理的测试。

赵志龙[34]设计了一种针对 Hadoop 的故障注入平台，这个平台可以对 Hadoop 的进程，节点，网络，分布式文件系统，资源占用进行故障的注入。缺点是只针对的是 Hadoop1.0 版本，而且没有对别的分布式平台进行测试。

1.3 本文结构

第一章是绪论部分，介绍了云计算故障注入平台的来源目的和意义，对云计算平台进行了简单的介绍，对基于软件的故障注入技术和云计算系统中的故障注入相关研究现状做了分析和调研。

第二章从云平台的层次结构入手，介绍了云平台的典型架构，深入分析了分布式计算平台中 Hadoop 和 Spark 这两种典型实例的体系结构，计算流程以及容错策略。虚拟化层中 Xen 和 KVM 这两种典型的虚拟化技术的特点和容错策略。云平台软件管理栈中的 CloudStack 的架构和容错策略。为后面故障注入平台的设计和各个层次故障注入工具的设计提供坚实的分析基础。

第三章介绍云计算系统故障注入平台的设计和具体实现。我们采用模块化的设计思想，尽量减少各个模块间的耦合度，并增加平台的可扩展性。

第四章，根据第二章的分析和第三章的总体故障注入框架的设计，在此基础上，我们对各个层次设计了故障注入工具模块。这些工具具有一定的易用性，能很好的测试云计算系统各个层次的容错性。

第五章是各个层次的故障注入实验和综合故障注入实验。对比层次内的实验结果和层次间的实验结果。为今后其它云平台的评测提供参考。

第2章 云计算系统的体系结构及容错机制研究

2.1 云计算整体架构

从服务模型的角度来定义云计算的话，我们会把云计算分为三层的服务模型，分别是基础设施层（IaaS），平台层（PaaS），软件服务层（SaaS），如图 2-1 所示。

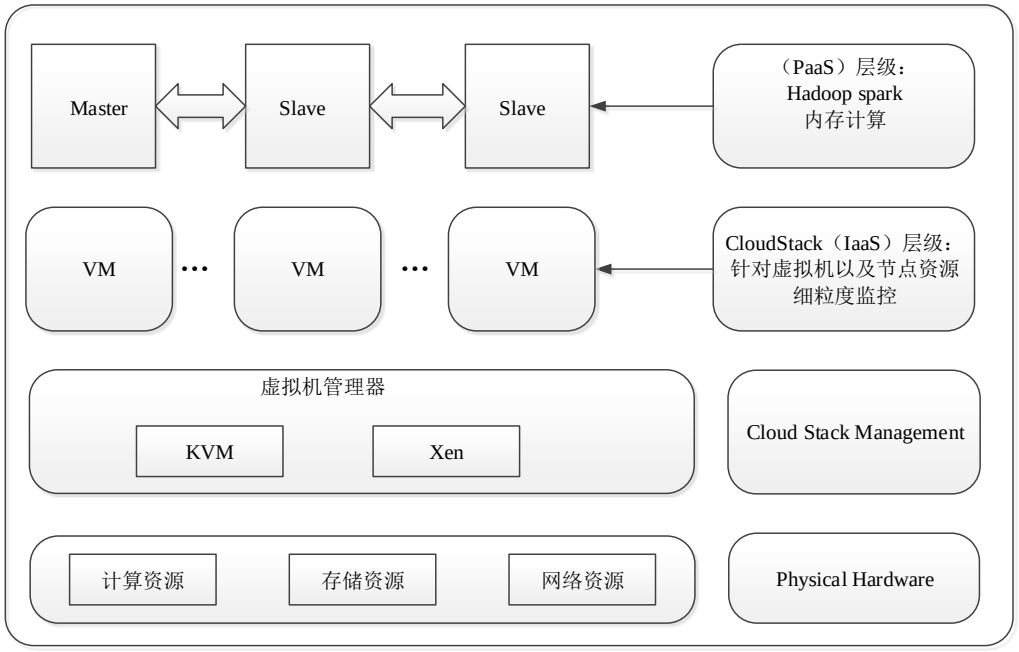


图 2-1 云计算架构图

IaaS 即基础设施即服务，它是一种最基础的云计算服务模型，它可以给用户提供基础计算设施，比如虚拟机，网络，存储等计算资源。这一层典型的技术包括虚拟化技术，例如 Xen， KVM，VirtualBox，Oracle VM 等，它可以把虚拟机当做访客一样处理。根据用户的需要动态的添加或者减少虚拟机。为了方便后续的研究，我们针对这一层选取了 Xen 和 KVM 两个开源虚拟化系统作为对虚拟化层的研究实例，并从中总结出共同的故障模型。以此为基础设计故障注入工具进行故障注入的测试。此外，为了方便的对所有的虚拟机，虚拟机管理器还有网络资源，存储资源计算资源等进行管理，人们设计出了云平台管理栈这个管理软件。我们选取了其中较为成熟的 CloudStack 作为这个层次的研究实例，并在后面进行了故障注入的实验。

PaaS, Platform as a Service，中文名为平台即服务。它可以给开发者提供便捷的开发平台，开发者不再需要自己配置开发的环境，这些工作都由云计算服务商提供。我们在这一层选取了两种分布式计算的平台来进行 PaaS 层容错性的

评测对象，它们分别是 Spark 和 Hadoop，这两种系统都是使用很广泛的计算框架，而且它们由很多共有的特性。这便于我们提炼共有的故障模型，而且便于我们进行对比分析。

SaaS，软件即服务。这是提供给一般用户的层次。作为普通的用户，接触到的就是这一层次，由于软件层的过于多样化，我们并没有在这一层选择具体的系统进行测试。

在本文中，为了对后面的故障注入工具的的设计和故障注入平台的设计部分进行铺垫。我们下面将对云计算平台的各个层次进行详细的分析，并对各个层次的典型架构进行研究，进而设计出针对该层次的较为通用的故障注入框架和工具和测试方法。下面我们将从分布式计算平台的架构和容错性，虚拟化系统的架构和容错性，云平台管理栈的架构和容错性这三方面进行研究和分析，进而设计出较为通用的自动化故障注入平台，使用该平台我们能够多层次的评测一个云计算系统的健壮性。

2.2 分布式计算框架架构研究和容错策略研究

典型的分布式计算框架包括 Hadoop、Spark、Storm[35]等，Hadoop 是开源社区根据 google 提出的 MapReduce 编程模型开展的一个项目，它现在已经成为了很成熟的的分布式计算框架。而 Spark 则是 UC Berkeley 提出的为了克服 Hadoop 磁盘 IO 较大对性能产生影响的缺点而提出的基于内存的计算模型。Storm 是一种实时的处理流式数据的分布式计算框架。它们既有很多相似的结构也有各自的不同，下面我们选择 Hadoop 和 Spark 进行具体的分析，为后面故障模型的设计做铺垫。

2.2.1 Hadoop 的架构和容错策略分析

如图 2-2 所示是 Hadoop 的架构图，由图我们可以发现 Hadoop 是一种主从的架构设计，有主节点 Master，还有从节点 Slave。在 Master 节点上运行有 JobTracker 进程，Namenode 进程。在 Slave 节点上运行有 TaskTracker 进程。JobTracker 负责任务的读取和分配。TaskTracker 负责任务的具体执行。具体的执行又分为 Map 进程和 Reduce 进程。各个进程间通过 RPC 进程通信。主节点和从节点之间还会通过心跳机制来检测从节点的进程和从节点是否正常运行。

Hadoop 执行一个具体任务的流程首先是 Client 端将一个作业提交到 JobTracker 端，然后 JobTracker 将作业放入一个作业调度器中然后进行初始化工作，将任务的输入进行划分，然后给每个划分生成一个 Map 任务，将 Map

任务分配到 TaskTracker 节点，当 Map 任务执行完后会进行 shuffle，然后生成 Reduce 任务，在 Reduce 任务完成后，本次的 job 会全部结束。通过对 Hadoop 执行流程的分析，本文可以提出针对一个 job 执行流程中的不同的进程和进程间的通信来注入故障，在 4.1 会有具体的故障集的设计。

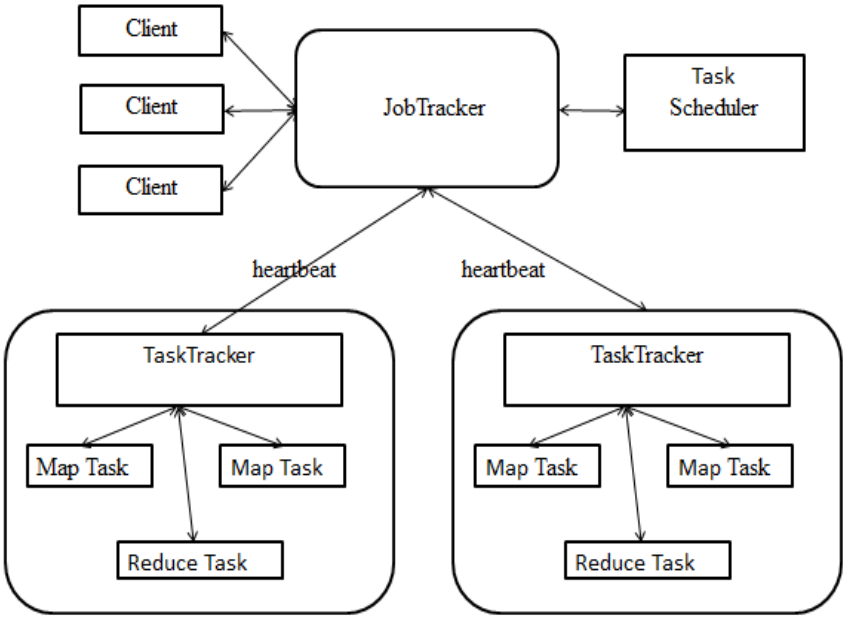


图 2-2 Hadoop 架构图

Hadoop 在设计上就考虑到了各种失败情况，所有节点之间通过心跳机制来检测各个进程是否正常运行。对于任务失败的情况，当 Map 或者 Reduce 进程异常结束时，Map 或 Reduce 任务中的代码会抛出运行时异常，子任务的 JVM 进程会在退出前将错误信息提交给 TaskTracker 进程，然后会通过重做机制启动新的任务来保证任务的正常执行。当 TaskTracker 进程失效时，它会停止向 JobTracker 发送心跳，JobTracker 从异常的心跳接收中会检测到 TaskTracker 已经异常，启动新的 TaskTracker 进程。而对于 JobTracker 失败，如果我们没有部署 ZooKeeper，那么整个 Hadoop 都会失效，如果我们提前部署了 ZooKeeper 那么我们可以通过提前部署多个 JobTracker 来防止这种单点故障的影响。

2.2.2 Spark 的架构和容错策略分析

如图 2-3 所示，从中可以看到 Spark 的各个组件间的简单的关系，分为 Client 节点，Master 节点和 Worker 节点。图 2-3 是 Spark 的作业执行流程图。与 Hadoop 执行的过程类似，都是由 Client 节点提交任务给 Master 节点，然后 Master 节点将任务分发给不同的 Worker 节点。有一个 Worker 会启动 Driver 来调度作业，

其余的 Worker 节点会启动 Executor 进程来执行具体的任务。Executor 在执行任务的时候是以线程为单位的，这是和 Hadoop 的不同之处。通过对 Spark 的任务执行流程的分析，我们提出了针对 Master 进程，Worker 进程，Executor 进程的故障注入，以及它们之间的通信状况的故障注入，具体的故障集的定义会在 4.1 介绍。

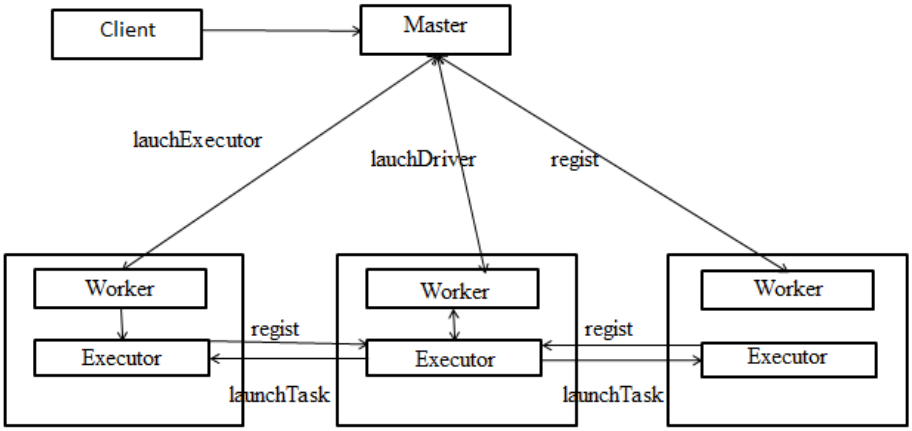


图 2-3 Spark 架构图

Spark 和 Hadoop 相比特点是使用了 RDD 这种自己设计的数据模型，它的独有的一些容错策略也是基于 RDD 的。Spark 选择记录更新的方式，但是只是粗粒度的，为了减小记录的成本，将 RDD 的相关操作记录下来。称为 Lineage。但是也有 doCheckPoint（检查点）方法，相当于通过冗余数据来缓存数据，本质是通过将 RDD 写入 Disk 做检查点，作为 Lineage 的辅助。

在通常的部署环境中，Spark 存在单节点故障，但是我们可以使用 ZooKeeper 来解决 master 节点的单点故障问题。Spark 本身提供了对 ZooKeeper 的支持。当我们配置好 ZooKeeper 后，Spark 会初始化多个 Master 进程，然后在 ZooKeeper 集群中注册各个 Master 进程，Master 进程的状态会在 ZooKeeper 中保存。其中一个 Master 会被选为激活的 Leader 模式，剩下的 Master 进程的会以 Standby 模式存在，不会向 Spark 提供服务，当 Leader 出现故障无法提供服务后，ZooKeeper 会检测到这种故障，然后在剩下的几个 Master 中选择一个新的 Leader 提供服务，并从 ZooKeeper 中读取状态恢复正常提供主节点服务的功能。

2.3 虚拟化平台架构研究

虚拟化平台可以将底层的硬件资源例如存储、网络、CPU 进行虚拟化后提供给上层使用。其中分为全虚拟化，半虚拟化技术。我们选取在开源领域两种

典型的虚拟化技术 Xen 和 KVM 作为本节的研究内容。

2.3.1 Xen 的架构

Xen 虚拟化系统是在硬件层之上享有最高权限的软件层[36],它由 Xen hypervisor 提供虚拟化支持。如图 2-4 所示，在 Xen hypervisor 之上，有一个或者多个客户操作系统可以寄存在上面。在 hypervisor 启动后，他会自动的装载第一个客户操作系统（Dom0），Dom0 是一个特殊的超级管理系统相比于别的虚拟机（DomU）而言，默认的情况下，Dom0 可以直接访问物理硬件资源。

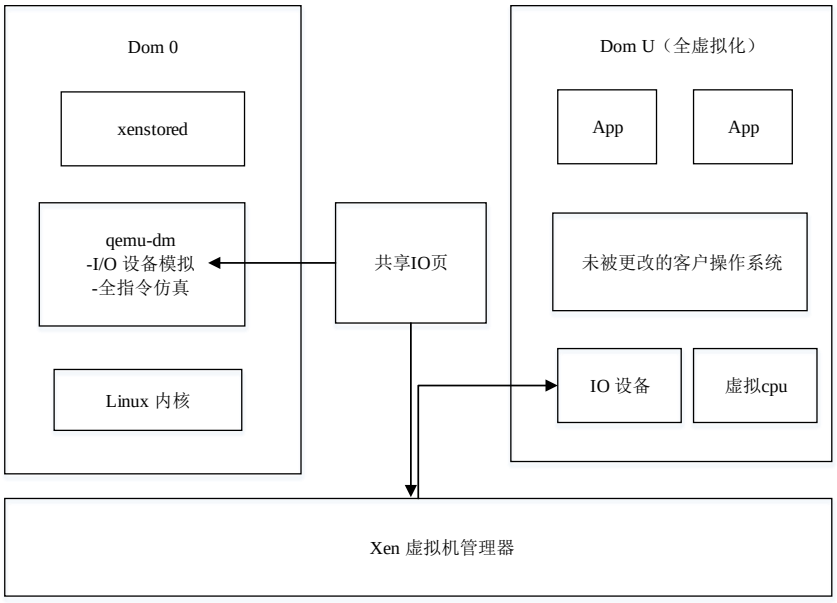


图 2-4 Xen 架构图

为了使原理和策略区分，Xen hypervisor 把控制接口交由 Dom0 来提供，即由 Dom0 负责具体的工作，应用层运行在 Dom0 上的管理软件利用 Dom0 提供的接口来管理系统提供的资源。例如 xenstored 就用来在虚拟机运行和创建 domU 设备期间存储关于 domains 的信息。为了支持没有被修改过的客户操作系统，Xen 同样使用了默认的 qemu（qemu-dm）来模拟虚拟硬件。和 KVM 类似，每个虚拟机都会有个一个 qemu-dm 进程运行在 Dom0。在本文后面，Xen hypersivor 和用户应用管理软件会是故障注入和分析的目标。

对于 Xen 而言，其中两个关键技术是超级调用和事件通道。我们在实现一些针对 Xen 特有的故障注入工具时，需要通过超级调用提供的接口和事件通道这两种方式来实现某些功能。

2.3.2 KVM 虚拟机管理器架构

KVM 是 Linux 虚拟机管理器，是基于内核的虚拟机管理器，可以在装载之后把 Linux 内核转变为虚拟机管理器。它通过内核模块的方式提供虚拟机化的功能，通过 KVM 我们让虚拟机像普通的 Linux 进程一样出现在 Linux 系统中。KVM 虚拟机管理器包含 KVM 内核模块和针对每个虚拟机的 qemu-kvm 用户进程模块。KVM 内核模块利用了由 x86 结构提供的硬件虚拟化（例如 Intel VT 和 AMD-V）来模拟虚拟机的 CPU。这种模块在进入客户模式控制虚拟机的内存管理时同样使用。在进入客户模式后，客户代码包括客户的系统和客户应用都是本地执行的，而不是仿真或者通过二进制翻译后执行的，直到它需要 IO 访问或者接受了中断。如图 2-5 所示，在 KVM 的体系结构中，所有的 IO 操作都会转发到用户模式下执行，通过硬件模拟器 qemu-kvm。一个 qemu-kvm 控制一个虚拟机的所有的 IO 访问，这是一个多线程的进程，会给每个虚拟机 CPU 创建一个线程，每个线程去模拟一个例如网卡的设备控制器或者磁盘控制器。

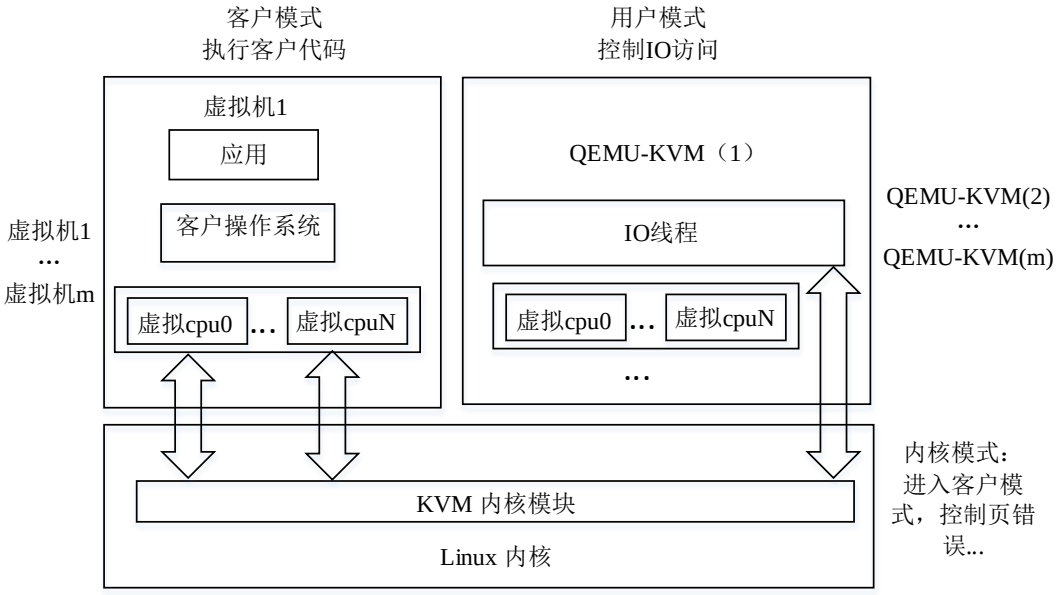


图 2-5 KVM 架构图

利用 intel 提供的 VT-x 虚拟机化技术的支持，KVM 虚拟机管理器中的每个客户虚拟机都能虚拟出多个虚拟处理器，即多个 VCPU。每个 VCPU 对应一个 qemu 线程，例如我们创建了一个拥有 4 个虚拟 CPU 的虚拟机，我们查看这个虚拟机对应的 qemu-kvm 进程，然后通过 pstree –p pid 查看这个进程对应的进程树。

图 2-6 所示为虚拟机对应的 qemu 进程，其进程号为 2505。如图 2-7 所示，2505 号进程有 4 个对应的线程这 4 个对应的线程是虚拟机对应的 4 个虚拟 CPU。

qemu-kvm 线程帮助虚拟 CPU 完成创建，初始化等功能。qemu-kvm 线程通过 ioctl 系统调用向/dev/kvm 传递对应的控制命令，然后内核模块再执行具体的创建虚拟 CPU 等具体的操作。通过/dev/kvm 这个设备，我们可以创建虚拟机，为虚拟机分配内存，向虚拟 CPU 读或者写寄存器，向虚拟 CPU 发出中断，运行一个虚拟 CPU。

```
[cs@kvm-centos ~]$ ps -ef | grep kvm
root       858     2  0 05:29 ?        00:00:00 [kvm-irqfd-clean]
avahi     1372     1  0 05:30 ?        00:00:00 avahi-daemon: running [kvm-cento
s.local]
qemu      2505     1 78 05:30 ?        00:00:18 /usr/libexec/qemu-kvm -S -M rhel
```

图 2- 6 qemu-kvm 进程

```
[cs@kvm-centos ~]$ pstree -p 2505
qemu-kvm(2505)─┬─{qemu-kvm}(2528)
               ├─{qemu-kvm}(2529)
               ├─{qemu-kvm}(2530)
               └─{qemu-kvm}(2531)
```

图 2-7 线程树

通过分析我们发现 ioctl 是一个很重要的函数，我们通过拦截 ioctl 函数加入自己定义的操作可以来模仿一些故障的情形。我们在后面的故障注入部分也有一部分基于 ioctl 函数来进行的，后面会详细叙述。

此外通过架构图的分析，我们可以对其中的虚拟 CPU，客户操作系统，Linux 内核模块，KVM 内核模块这几个部分进行故障注入。

2.4 云平台管理栈架构研究和容错策略研究

本节我们选取的研究实例是 CloudStack 这个云平台管理栈，和 OpenStack，Eucalyptus 等管理栈相比，它的优点是部署方便，用户 UI 接口简单易用，已经被很多的企业使用。缺点是容错机制还不是很完善，目前开源社区在 CloudStack 领域不是很活跃。我们下面将从 CloudStack 入手总结出云平台管理栈的一些典型特点，为这个层次的故障注入做准备。

2.4.1 CloudStack 架构研究

CloudStack 的架构是主从架构，最简单的部署情况下有一个管理节点，一个计算节点和一个存储节点，如图 2-8 所示。在高级的部署架构中可以部署多个管理节点，计算节点和存储节点来实现负载的均衡和容错。管理节点即管理服务器，它可以部署在虚拟节点上，也可以部署在物理节点上，它是 CloudStack 云管理平台的关键组件，主要将整个 IaaS 平台的所有管理工作汇总到这个核心

来处理，负责分配和安排整个云平台的资源，接受用户和管理员的操作，包括对硬件、虚拟机和网络的管理和维护，管理节点会将用户通过 web UI 或者 API 的调用命令转化为各个物理资源可以接受的命令，然后按照它们定义好的通信格式，向计算节点发送各种命令，可以创建虚拟机，删除虚拟机，创建虚拟机模板，迁移虚拟机。向存储节点发送命令，可以完成主存储或者二级存储任务。管理服务器运行在包含一个 Apache Tomcat 容器中，而且需要 MYSQL 数据库作为持久化的需要。通过 MySQL，CloudStack 可以将计算节点，存储节点，还有虚拟机的状态，网络资源的分配和使用情况保存下来供管理人员管理和查询使用。虚拟化服务器集群是计算节点组成的集群，用来部署虚拟机实例。存储部分用来部署主存储和二级存储。

图 2-8 CloudStack 架构图

图 2-9 展示了较为详细的结构，在系统启动后，会在第一个计算节点上创建两个系统虚拟机，一个是二级存储虚拟机，负责管理镜像，模板快照等存储资源，还有一个控制台代理虚拟机，负责在网页端来给用户提供命令行的窗口。此外还有一个虚拟路由器，可以给平台提供 dhcp 等网络服务。我们可以看到在一个节点上还有主存储和二级存储服务器。主存储保存了在计算节点上运行的虚拟机实例的虚拟磁盘，它可以设置为本地存储或者网络存储等存储方式。网络存储的优点是可以在虚拟机故障时可以在不关闭虚拟机的情况下实现虚拟机的迁移，此外还可以开启高可用（HA）特性。而选择本地存储为主存储的实现方式的优点是，虚拟机的磁盘读写有很好的性能，不需要较高的网络带宽，但是无法实现虚拟机的在线迁移。二级存储可以用来保存系统镜像的模板，ISO 启动镜像还有磁盘卷的快照，它是作为主存储的辅助存储而提出的，它的存在可以让我们便捷的存储快照、镜像、模板等存储资源。

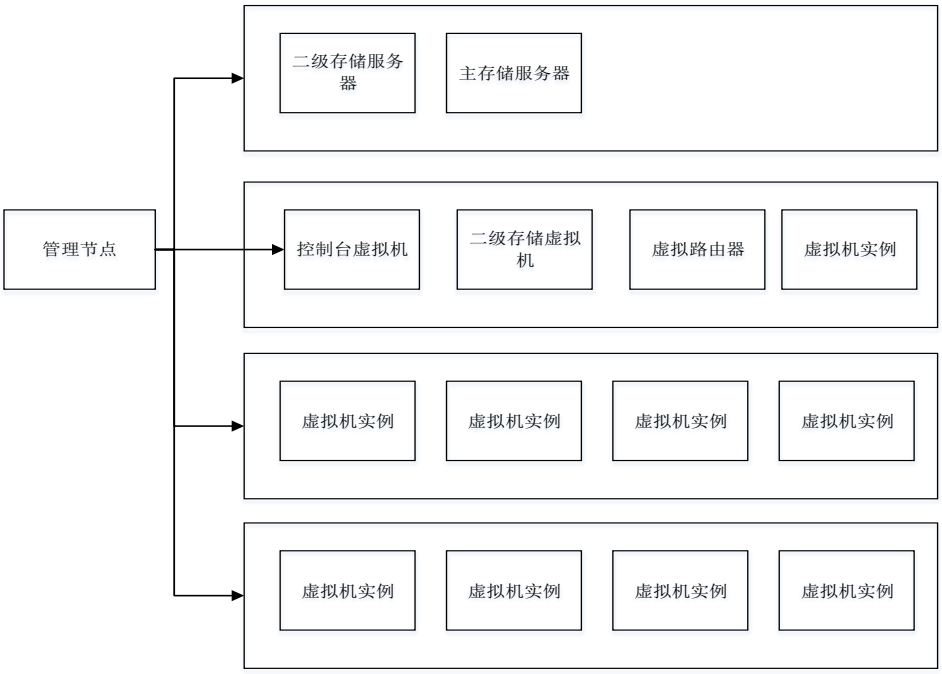


图 2-9 细化的 CloudStack 架构

2.4.2 CloudStack 容错策略研究

CloudStack 的容错机制采用了重做的思想，将失败的任务进行重启。

具体来说，CloudStack 本身对系统虚拟机提供了容错的策略，当系统虚拟机异常关闭的时候，管理节点会检测到这一行为然后重新启动新的对应的系统虚拟机。此外 CloudStack 还提供了针对虚拟机的高可用性和计算节点的高可用性，开启这两个功能后，异常关闭的虚拟机实例会自动重启，计算节点损坏后，该计算节点上的虚拟机实例会自动迁移到其余的计算节点上。

2.5 本章小结

本章简要分析了云平台的典型的架构，从分布式计算平台，虚拟化平台，云平台管理栈这 3 个层次入手分析具体的架构、容错机制和策略。本章为第四章各个层次的故障注入模块的设计做了很好的铺垫。下一章将介绍整体的故障注入的框架设计和实现。

第3章 云计算系统故障注入平台框架研究与设计

云计算平台作为一个较复杂的整体，其中包含较多的组件，如果用手工的方式进行故障注入的测试将会有很多重复性的工作，而且效率极低，为了提高测试人员的测试效率以及易用性，本文设计了这套针对云计算系统的故障注入框架。该框架可以将测试人员从繁琐的测试工作中解脱出来，专注于测试的方法和结果本身，提高测试的效率。

3.1 云计算系统故障注入平台整体设计

本文设计的故障注入平台针对完整的云计算系统，并且旨在能够对不同的云计算系统均能够实现方便的故障注入。采用本文设计的框架，可以根据云计算系统的各个层次组件的不同，故障注入模块也可以实现方便的替换。从而能够解决对云平台进行故障注入时各种复杂问题的出现。

在以往的工作中，有单独对虚拟机管理器进行故障注入的，有对分布式计算平台进行故障注入的，有对云平台管理软件进行故障注入的，但是没有对一个完整的由这些组件搭建的一个多层次的云计算系统通过故障注入的方式进行系统的健壮性测试的。而且也缺乏一个自动化得平台能达到此目的，本文就是针对这两点尽可能的来构建一个统一的针对云计算系统的能够进行多层次的故障注入测试的自动化测试平台。

本平台的设计目的是能够方便的对云计算平台的各个层次在统一的框架中做到自动化测试。而且本平台是可扩展的，根据云系统中组件的增添，故障注入平台中的故障注入的组件也可以实现方便的增添,从而方便今后的测试工作。本平台以模块化形式组成，由以下几个部分组成，包括：自动部署模块，故障生成模块，故障控制与注入模块，故障检测与系统监控模块，负载发生模块，结果收集与分析模块，数据库模块。各个模块之间尽量相互独立，各自完成较为独立的功能，这方便了今后的扩展的工作。一个典型的云平台中包含 IaaS，PaaS,SaaS 这几个层次，我们的平台最终可以做到对这几个层次能够做到自动化的健壮性测试。

3.1.1 故障注入平台的框架

故障注入平台的设计如图 3-1 所示，故障注入部分是核心功能，其余模块起到辅助的作用。测试的主要步骤：（1）自动部署模块根据配置文件部署好各个节点的环境。（2）我们通过故障控制模块生成实时的输入或者脚本的输入。

（3）将故障分发到各个故障注入器，故障注入器在合适的时间调用故障注入模块进行故障注入。（4）将注入的结果进行回收，分析统计信息。

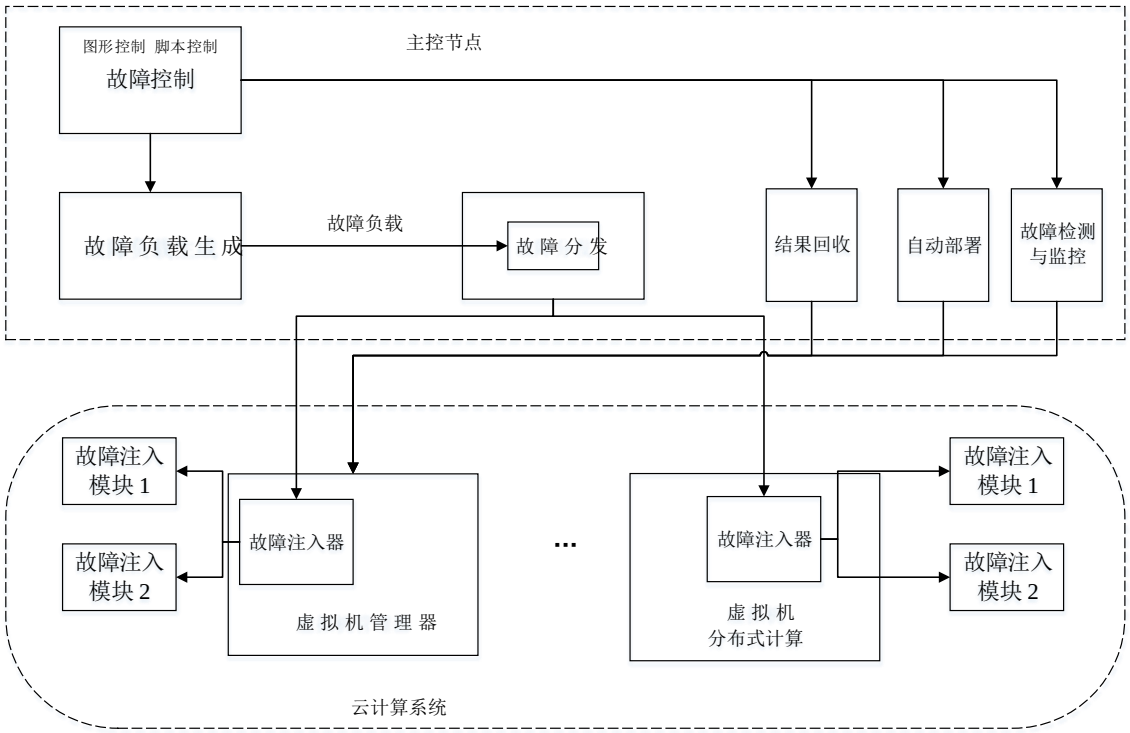


图 3-1 故障注入平台架构图

故障控制模块可以将测试人员需要生成的故障通过图形或者脚本的方式进行控制，生成故障负载，然后控制节点根据故障负载的 ip 信息分发到各个具体的物理节点或者虚拟机节点的故障注入器。故障注入器会控制在特定的时间点注入故障。结果回收模块将把故障的注入情况存储在数据库中，进行后续的统计分析。

每个模块在控制端和故障注入端都有对应的守护进程，控制端的守护进程用来和用户交互，向故障注入端发送该模块的具体的命令。故障注入端的守护进程来完成具体的工作，例如调用故障注入工具进行故障注入，进行系统监控，故障检测等。

3.1.2 故障注入的层次设计

图 3-2 是本文设计的针对云计算系统故障注入的层次，我们将典型的故障注入的层次分为 3 层，分别是分布式计算平台故障注入，虚拟化故障注入和云管理软件栈故障注入。这 3 个层次的故障注入分别对应 3 大类故障注入工具。

这三大类的工具提供统一的接口，可以由各个节点的故障注入器进行调用。这三大类工具的具体设计和实现在第四章详细介绍。

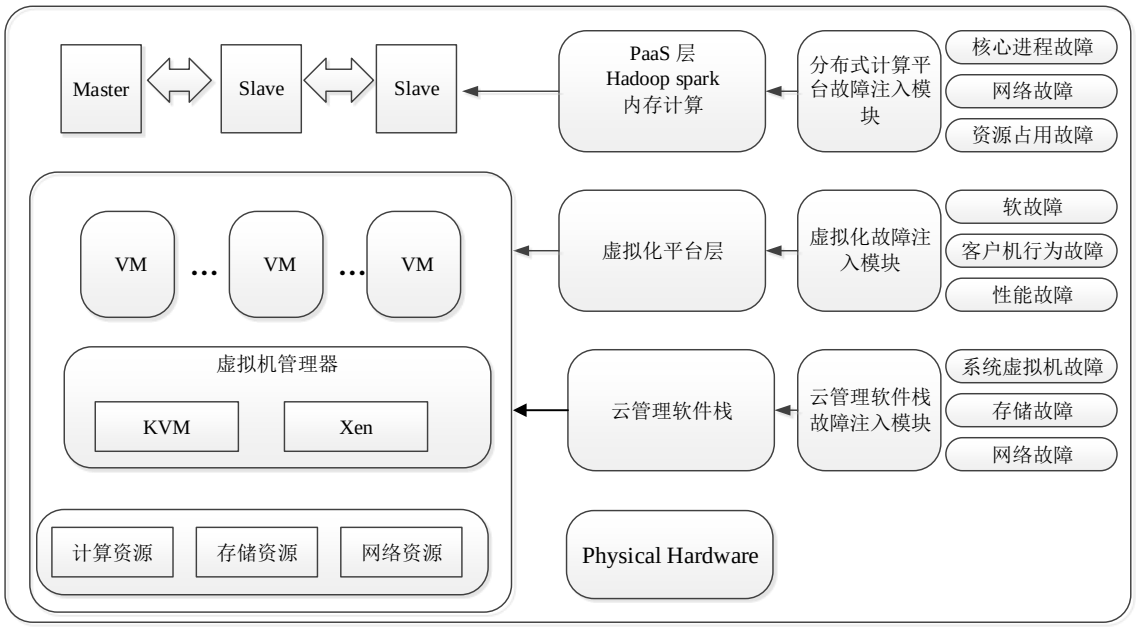


图 3-2 故障注入的层次

3.2 各个模块的具体实现

3.2.1 自动部署模块

本平台可以在云环境中自动的部署故障注入的环境，集群中的各个节点的 ip 地址会在配置文件中以参数的形式进行设置，其中一个节点作为总控节点，我们首先将所有的文件安装到总控节点后，在配置文件中配置好哪些节点需要部署故障注入的脚本，哪个节点来部署数据库服务，哪个节点来收集数据，配置好之后，我们的自动部署程序就可以把对应的文件分发到对应的节点。

```
{
    "autodeploy":[
        {"ip":"192.168.10.2"}
        {"ip":"192.168.10.3"}
        {"ip":"192.168.10.4"}
    ]
    "hostnode":{"ip":192.168.10.1}
    "databasenode":{"ip":192.168.10.20}
}
```

图 3-3 配置文件样例图

本文采用 json 的格式来定义我们的配置文件，如图 3-3 所示，在配置文件中定义好了自动部署的节点为 3 个，主节点为 192.168.10.1，数据库节点为

192.168.10.20。此外因为我们是基于云环境下的测试工作，我们的虚拟机节点可以方便的进行扩展，我们在实验中使用的是 CloudStack，在配置好一个虚拟机节点后，我们可以将该节点制作成模板，然后快速的生成多个虚拟机节点，供我们故障注入的实验使用，这可以很大程提高环境的部署效率。

3.2.2 故障生成模块

根据对典型的云计算系统的分析，本文设计了如下故障模型，即故障层次，故障工具，故障位置，故障时间，故障参数。这是一个 5 元组，我们注入的每一个故障都可以用这一个 5 元组进行刻画，首先按照这种格式我们把后面的各个层次的故障注入工具进行整理如下。表 3-1 是分布式平台的故障模型，表 3-2 是虚拟化层的故障模型，表 3-3 是云管理栈的故障模型。使用这种格式，我们可以统一的定义各个层次的故障，这位我们后面的故障注入提供方便。

表 3-1 分布式计算平台层故障模型

故障层次	故障工具	故障位置	故障发生时间	故障参数
分布式计算平台（Hadoop/Spark）	进程故障	ip 地址	xx:xx:xx	进程名称
分布式计算平台（Hadoop/Spark）	网络故障	ip 地址	xx:xx:xx	延迟\丢包\乱序
分布式计算平台（Hadoop/Spark）	资源占用	ip 地址	xx:xx:xx	CPU\内存

表 3-2 虚拟化层故障模型

故障层次	故障工具	故障位置	故障发生时间	故障参数
虚拟化层（KVM Xen）	软故障	ip 地址	xx:xx:xx	寄存器名称
虚拟化层（KVM Xen）	客户操作系统故障	ip 地址	xx:xx:xx	错误表现类型
虚拟化层（KVM Xen）	性能故障	ip 地址	xx:xx:xx	延迟时间
虚拟化层（KVM Xen）	维护故障	ip 地址	xx:xx:xx	CPU 序号

表 3-3 云平台管理栈故障模型

故障层次	故障工具名称	故障位置	故障发生时间	故障参数
CloudStack	存储故障（主存储、二级存储）	ip 地址	xx:xx:xx	读/写
CloudStack	系统虚拟机故障	ip 地址	xx:xx:xx	控制台虚拟机、二级存储虚拟机、虚拟路由器
CloudStack	网络故障	ip 地址	xx:xx:xx	延迟\丢包\乱序
CloudsStack	虚拟机操作故障	ip 地址	xx:xx:xx	创建、迁移、资源分配
CloudStack	Management 内存占用故障	ip 地址	xx:xx:xx	内存占用率

我们首先根据针对云计算平台的故障注入的层次将所有的故障模型分成了以上 3 大类，分别是分布式计算层，虚拟化层，云平台管理栈层。分布式计算层定义了针对分布式计算层面的故障模型，虚拟化层定义了针对虚拟机以及虚拟机管理器的故障模型，云平台管理栈定义了针对云平台管理栈的故障模型。当具体到某一个具体平台实例时我们可以通过对故障少量修改达到更高的针对性。如果需要增加故障注入的模块时，我们只需要根据上面的格式定义好接口，然后就可以集成到我们的故障注入平台中。

利用上面我们定义的故障模型，我们可以方便的定义一个要注入的故障，并且便于分析和收集。比如我们要注入 Hadoop 层次的上的一个 Map 进程故障，那么我们可以用以下的（Hadoop，进程故障注入，192.168.16.10，12:40:42，map）这个 5 元组来定义。XenServer 的 CPU 故障我们可以用（Xenserver， CPU 故障， 192.168.10.196， 11:22:11.，ip 1bit，2 分钟一次）类似来定义。CloudStack 的二级存储故障可通过类似（CloudStack， 二级存储故障， 192.168.10.196，11:22:21，一天 4 次 ）的 5 元组来定义。我们可以看到各个层次的故障的格式是统一的，这就为后面的工作提供了方便。我们即可以手工的编写故障注入的配置文件来定义在预定的时刻，我们让故障以什么样的分布来发生，也可以在图形控制界面中实时的实现故障的注入，或者说通过故障配置文件的定义通过笛卡尔积来自动生成故障集，这需要看具体的需求场景来确定。

3.2.3 故障控制与注入模块

这个模块是系统的核心部分，故障的控制部分在总控节点，注入模块在各个节点上，总控端可以读取 2 种形式的输入，一种是通过读取故障注入脚本的输入，在给定的时间往预定的位置注入故障。一种是根据需要在需要的时间点，通过图形界面的接口手动进行故障注入。图 3-5 所示为故障输入的流程图。无论是通过故障注入脚本还是通过图形界面输入的故障注入的命令，它们都会被经过控制节点处理，控制节点把两种方式得到的故障注入的输入，会根据输入中的 ip 地址将该请求元组分发到对应的故障注入节点，每个故障注入节点都有一个守护进程在不停的运行，守护进程会维护一个消息队列，守护进程不停地的检测该队列头，当队列头中的故障注入任务满足发生的条件时，守护进程会将该任务从队列头中取出，调用对应的故障注入模块，并传递相应的参数，然后故障注入程序运行，依次循环，直到队列为空。故障注入模块在第四章中被分为 3 个层次，这三个层次的模块具有相同的接口。故障注入器通过调用对应的故障注入模块来实现最终的故障注入的功能。

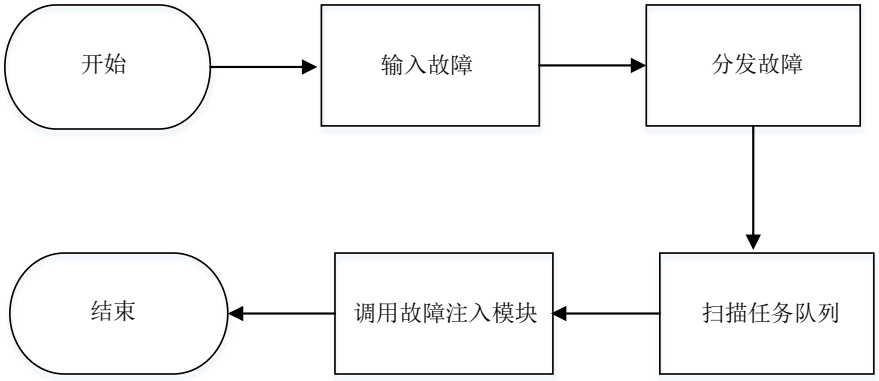


图 3-5 故障输入流程图

3.2.4 故障检测与监控模块

本模块起到的是辅助作用，云计算系统本身在各个层次都提供了错误检测机制，本模块只是额外的监控功能，可以检测到一些明显的故障类型。

检测网络故障,通过周期性的向预先设置的节点发送 ping 命令，如果在给定的时间范围内能正常返回结果说明该节点正常,反之说明该节点出现网络故障。

检测进程故障，我们在配置文件中配置好需要检测的进程名称，然后周期性的向各个节点发送查询对应的守护进程是否正常运行的命令，正常运行的进程的名称提前定义在配置文件中，将返回的进程名称和配置文件中的名称进行对照，若返回的进程名不完全包括配置文件中的进程名就说明出现进程故障。

从日志中检测：这需要在每个节点上的守护进程上实现，在每次注入一个故障后，去日志中扫描在故障注入后的时间点到对应的时间窗口的的日志信息，查看其中是否有出错的相关信息的关键词出现。出现说明系统检测到故障。

而系统监控主要是在守护进程中定时检测 CPU 和内存利用率等基本信息。旨在通过系统监控发现明显的系统问题，如果发现异常信息，就会发送给总控节点。

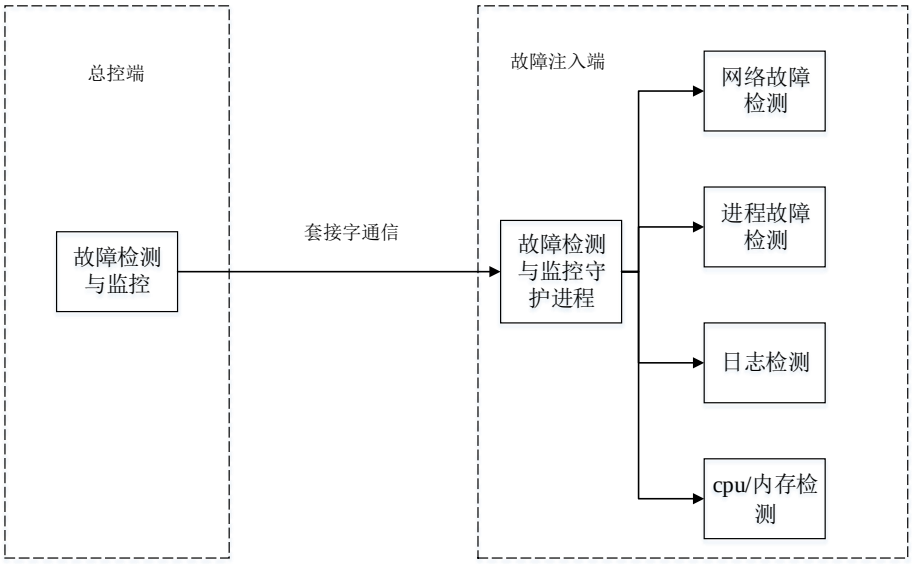


图 3-6 故障检测与监控架构图

3.2.5 数据库模块

数据库模块负责存储整个故障注入过程中所有的操作信息，比如各个时间点注入的各种故障的信息，系统的崩溃的信息，负载的运行后的统计结果额信息。通过对故障注入平台的所有相关的操作的记录，我们可以在之后的统计和分析中得到更全面的分析结果。我们定义如下几个表，故障注入记录表，系统运行负载表和系统故障检测表。

3.2.6 结果收集与分析模块

将数据库中存储的相关历史信息，通过各种统计手段，计算失效率，无故障运行时间，故障前后的性能对比等结果，生成各种图表，提供可视化的结果。比如我们在故障注入表中记录都在什么时刻针对哪个层次注入了什么故障，使用的故障注入工具是什么。

3.2.7 回滚模块

在以往的故障注入实验中由于故障注入的种类多种多样，有些故障在系统重启后能恢复正常，而有些故障在系统重启后是无法恢复的。这就需要通过重做系统等手段来使系统恢复到正常状态。而这种方式的成本是很高的。因此，为了克服这种缺点。对于在虚拟机上的故障注入，我们设计了回滚模块，通过虚拟机快照的方式，我们使得系统可以方便的恢复到故障注入之前的状态，从而提高故障注入的效率。

在每次故障注入之前，测试人员可以通过该模块选定需要对那些节点制作快照，在执行完一次故障注入的实验后，测试人员可以通过该将所有的虚拟机节点恢复到之前没有注入故障时的状态，从而消除对下一次注入测试的影响。具体实现上因为我们使用 的 CloudStack 管理平台，我们可以通过 CloudStack 提供的 API，调用相关的虚拟机快照和和回滚的操作来实现相关的功能，或者通过 CloudStack 提供的图形接口来进行快照回滚的操作。

3.2.8 负载管理模块

该模块负责管理各个层次的各个负载，方便测试人员运行各种负载时需要输入繁多的配置参数，通过该管理模块，测试人员可以方便的将负载部署到对应的节点，并监控该负载是否在正常的运行状态。通过该模块的配置文件，测试人员可以从重复性的负载命令中解脱出来，方便的调用各种历史负载命令，并保存负载的运行信息，将负载执行情况保存到数据库中，并在数据库中查看各个历史负载在故障注入的情况下的运行情况，为后面的分析工作做铺垫。目前本模块集成了 Hadoop 负载管理，Spark 负载管理，虚拟机负载管理，CloudStack 负载管理这几部分，随着测试人员的需要，可以按照制定的格式，添加或者删除一些配置从而达到负载的增添与删除的目的。

3.3 本章小结

本章详细介绍了云计算系统故障注入框架的设计，对其中的各个模块的功能进行了划分，对自动部署模块，故障生成模块，故障控制与注入模块，故障检测与监控模块，数据库模块，结果收集与分析模块，回滚模块的流程进行了详细的设计，后面将对集成的各个故障注入模块进行研究和设计。

第4章 多层次故障注入工具的设计与实现

本章将会从分布式计算层，虚拟化层，云平台管理栈三个层次对各个故障注入工具进行设计与实现，这三个层次的故障注入工具是由第三章中故障注入器模块进行调用的。

4.1 多层次故障注入工具的设计

4.1.1 分布式计算平台故障注入工具的设计

在分布式计算平台可以看做是 PaaS 层，因为在分布式计算平台上，用户可以通过分布式计算平台提供的编程模型，编写程序，然后在平台上运行。我们选取了两种典型的分布式计算平台 Hadoop 和 Spark 作为这个层次容错测试和研究的例子。

Hadoop 和 Spark 的共同点是都是为了进行分布式计算而设计的，他们都可以减少编程人员进行分布式编程时的负担，他们为编程人员隐藏了分布式平台下复杂的通信机制，容错机制，使用平台提供的简单的接口，我们就可以编写在平台上运行的复杂的程序。为了对分布式平台这一层进行容错性的可用性的测试，我们选取了 Hadoop 和 Spark 这两个典型的平台来进行实验和对比。

通过对 Hadoop 和 Spark 的体系结构进行了分析，我们发现，作为一种分布式的计算框架，它的很多的服务运行在了不同的节点上，而且不同的节点有主次节点之分。主节点是 Master 节点，从节点是 Slave 节点，主节点负责任务的调度和分配，而从节点负责任务的具体的执行过程。因此，从这个角度来看可以分别向主节点和从节点注入故障，因为主节点和从节点在系统中的重要性不同因此故障注入后，系统表现出反应也是不同的。因为主节点负责总体的控制流程，当主节点发生故障后，系统会发生较为严重的故障，例如在执行 job 过程中，主节点网络延迟过大，这会使得主节点和各个从节点间的通信都发生很大问题，这个时候控制程序会认为几乎从节点已经执行过的某些 Map 任务或者 Reduce 任务出错，此时系统会认为发生了重大故障，job 会异常停止。但如果只是在单个的从节点注入了部分的网络故障，例如延迟故障，系统会将那个节点上已经执行过的任务迁移到其余的可用节点，采用这种重做的容错机制，系统可以容忍某些故障。因此，我们将具体从网络，节点，进程，文件系统这几个维度来进行故障的注入和分析，最后会将系统进行集成测试。

4.1.1.1 核心进程故障的设计

在进程的维度， Hadoop（Spark）的各项服务都以进程的形式进行提供，在主节点和从节点都运行了一些守护进程。这些守护进程构成了 Hadoop（Spark）的各项关键服务。当提交一个任务时，在这个守护进程的基础上，在不同的节点根据需要会启动相应的工作进程。从重要性的角度来看，守护进程的重要程度肯定重于工作时启动的进程。从故障注入的角度，我们可以分别对守护进程和工作时进程这两个方面对进程进行分类。对于 Hadoop 而言，Map 进程和 Reduce 进程属于提交任务后的进程，而像 NameNode，Secondary NameNode，JobTracker，TaskerTracker，DataNode 属于守护进程。对于 Spark，CoarseGrainedExecutorBackend 进程属于提交任务后的进程，而像 Master，Worker 属于守护进程。Hadoop 核心进程故障集定义如表 4-1 所示。对于 Spark 的核心进程故障集只需要根据进程的名称替换 Hadoop 中的进程名称即可如表 4-2 所示。

表 4-1 Hadoop 核心进程故障集

故障名称	故障描述	故障参数	预期效果
Map 进程故障	Map 进程意外终止	ip 地址	Map 进程意外终止
Reduce 进程故障	Reduce 进程意外终止	ip 地址	Reduce 进程意外终止
TaskTracker 进程故障	TaskTracker 进程意外终止	ip 地址	TaskTracker 进程意外终止
JobTracker 进程故障	JobTracker 进程意外终止	ip 地址	JobTracker 进程意外终止

表 4-2 Spark 核心进程故障集

故障名称	故障描述	故障参数	预期效果
Master 进程故障	Master 进程意外终止	ip 地址	Master 进程意外终止
Worker 进程故障	Worker 进程意外终止	ip 地址	Worker 进程意外终止
CoarseGrainedExecutorBackend 进程故障	CoarseGrainedExecutorBackend 进程意外终止	ip 地址	CoarseGrainedExecutorBackend 进程意外终止

从服务的角度来看，使进程失效最简便的方法就是使用 kill 命令，对于守护进程因为每个节点没有重复的，所以先通过 jps -m 命令查看进程名对应进程

号然后 kill 掉即可。这种方式只能模拟进程的异常结束，通过 kprobes 技术，我们可以在用户态跟踪一个进程，然后利用 kprobes 提供的接口，实现对进程的寄存器或者数据段，代码段等的修改，模拟进程的不同种类的异常情况。对于有重复进程的 map 和 reduce 返回相关的进程号好随机的在其中选择一个或者几个所需的进程然后 kill。由于 Hadoop 存在 1.0 和 2.0 版本，在这两个版本中，系统中出现的进程和进程信息有一些不同，但由于针对进程的故障注入的原理是相似的，所以我们同时实现了对 1.0 和 2.0 两个版本的进程的故障注入。

4.1.1.2 网络故障注入模块的设计

对于分布式计算平台而言，网络通信是其中很重要的一环，分布式系统上各个进程分布在不同的节点上，各个节点上的进程需要通过网络来进行通信，当网络出现故障时，一定会对分布式系统的通信产生影响，因此分布式系统在设计之初就应该考虑到对各种网络状况的异常处理。本模块就是来模拟可能出现的各种网络故障。

具体到我们研究的两个具体的计算平台 Spark 和 Hadoop 而言，根据它们提供的进程和节点的种类，我们可以用笛卡尔积的方式定义出所有可能出现的进程和进程间，节点和节点间的网络故障集。

在网络的维度，网络负责进程的跨节点的通信。本质上来说，网络的故障将导致进程的故障，但是这是两个不同的角度，一个节点上的网络故障会导致这个节点上几乎所有的需要和外界进行通信的进程都受到影响，因此网络的故障造成的影响的范围更大。因此从这个方面考虑确实很重要。对于网络的故障注入可以采用 Linux 内核提供的 tc 命令工具集来实现简单的延迟，丢包等网络故障的模拟。如果要实现具体到某几个进程的网络故障这就需要使用 Linux 内核提供 netfilter 工具自己进行编程来实现。网络故障集定义如表 4-3 所示。

表 4-3 网络故障集

故障名称	故障描述	故障参数	预期效果
主节点和从节点间故障	主节点和从节点之间通信出现故障	ip 地址，丢包，延迟	影响进程正常通信
从节点间网络故障	从节点间通信出现问题	ip 地址，丢包，延迟	影响进程正常通信

从节点这个维度来考虑，节点的故障和严重的网络通信故障等同。因此，我们不再单独对节点宕机的故障做单独的测试。

4.1.1.3 资源占用故障模块的设计

资源占用故障描述的各个节点在计算资源占用率过高时系统故障情况。我

们在此设计了内存和 CPU 两种类型的计算资源的占用种类。根据 Spark 和 Hadoop 节点种类的不同，我们分别总结如表 4-4。

表 4-4 资源占用故障模型

故障名称	故障描述	故障参数	预期效果
主节点内存占用故障	主节点节点内存占用过高	ip 地址、占用率	影响主节点性能
从节点内存占用故障	从节点节点内存占用过高	ip 地址、占用率	影响从节点性能
主节点 CPU 占用故障	主节点节点 CPU 占用过高	ip 地址、占用率	影响主节点计算能力
从节点 CPU 占用故障	从节点节点 CPU 占用过高	ip 地址、占用率	影响从节点计算能力

4.1.2 虚拟化层的故障注入工具的设计

4.1.2.1 软错误

这个故障模型模拟了发生在内存，寄存器，或者 CPU 的执行或者控制单元中的软错误或者瞬时故障。这个故障的代表是一位或者多位翻转。软故障时一种众所周知的现象，它是由于半导体设备由于外部的粒子辐射和系统运行过程中的功率峰值所引发，由于误码率的增加这个问题会变得恶化。各种类型的软错误的如表 4-5 所示。

表 4-5 软错误故障模型

故障类型	故障描述	例子
一位翻转	一位或者多个位的翻转	0x10 到 0x11
交换	两位交换	0x10 到 0x01
覆盖	对一个目标寄存器或内存写入特殊的值	把 0x00000001 写入目标内存
NOP	将 NOP 指令写入代码段	原始的值是 0xabceef20，新的值是 0x90909090

4.1.2.2 客户操作系统的错误行为

这个故障模型模拟了客户操作系统的错误行为进而导致了系统状态失控或者 CPU 异常，最后这个不寻常的行为可能使客户操作系统崩溃或者宕机。这个故障模型使用了黑盒测试的模型的方法来测试客户操作系统和宿主系统间的的故障的隔离性，即是否故障会从客户操作系统传递到宿主操作系统，这对一个

虚拟机系统来说是很重要的。在理想情况下，所有的故障情况都应该在客户操作系统上进行确认。我们通过在 KVM 和 Xen 的客户虚拟机中注如错误行为的故障来评测虚拟机管理器对这种故障类型的容错能力。

4.1.2.3 性能故障

这个故障模仿了一个或者多个线程由于 IO 访问的阻塞或者 CPU 的耗尽而出现延迟的情况。虚拟机背后的核心思想是去抽象一个物理机的硬件来使客户操作系统来共享。然而如果这些共享的资源不能够合适的同步，那么这可能会导致竞争条件，这可能会最终导致系统出现意料之外的行为。除此之外，由于问题的时间的独立性和不确定性，检测这种 bug 是很困难的，尽管它很重要。

因为竞争条件是时间敏感的，因此它很有可能在高负载的情况下出现。例如，一种竞争条件的 bug 是检测的时间到使用的时间时（TOCTTOU）的缺陷，TOCTTOU 是指计算机系统的资料与权限等状态的检查与使用之间，因为某个特定状态在这段时间已经改变所产生的软件漏洞。这种缺陷很有可能发生如果 TOCTTOU 被延长。因此，这个故障模型的设计是用来检测虚拟机管理器在多个线程在某个特定的时间被锁住一段时间时的实现是否出现问题。

4.1.2.4 维护故障

这个故障模拟了某些特定的硬件需要被关闭来进行更换或者是电源管理。这个故障用来评估虚拟机管理器的维护能力和易管理能力。现代的系统的架构和当前版本的 Linux 内核支持 CPU 的热插播在在线维护的时候，根据需要升级容量，节省能源而且不会影响可靠性。对于云组件来说这个特性很重要。我们通过对 CPU 核心的热插拔来模拟维护故障。

4.1.3 云平台管理软件栈的故障注入工具的设计

我们根据 CloudStack 的架构和特点可以归纳出一下几种故障类型，分别是存储故障，系统虚拟机故障，网络故障，虚拟机操作故障，管理节点资源故障。这几种类型包含了 CloudStack 在运行和维护的过程中可能出现的一些故障情形。

4.1.3.1 存储故障

CloudStack 的存储分为主存储和二级存储，主存储还分为网络存储和本地存储，我们通过存储故障这个故障类型来模拟 CloudStack 在实际使用过程中可能出现的二级存储或者主存储故障，主存储故障会影响到系统磁盘的读写，二级存储故障会影响到 ISO 和模板的读写。我们通过此故障来评测 CloudStack 对存储故障的容错能力。在具体的实现原理方面，我们采用了底层的文件系统的

故障注入原理的技术，将二级存储或者主存储对应的本地目的作为故障注入模块的参数，通过底层的读或者写的错误来模拟上层的二级存储或者本地存储的错误。存储故障集如表 4-6 所示。分别包含二级存储故障、主存储故障和 NFS 存储故障。

表 4-6 存储故障模型

故障名称	故障描述	故障参数	预期效果
二级存储读失效	无法从二级存储正常读数据	ip 地址，二级存储目录	无法正常读
二级存储写失效	无法从二级存储正常写数据	ip 地址，二级存储目录	无法正常写
本地主存储读失效	无法从主存储正常读数据	ip 地址	无法正常读
本地主存储写失效	无法从主存储正常写数据	ip 地址	无法正常写
NFS 主存储读失效	无法从主存储正常读数据	ip 地址，主存储目录	无法正常读
NFS 主存储写失效	无法从主存储正常写数据	ip 地址，主存储目录	无法正常写

4.1.3.2 系统虚拟机故障

系统虚拟机在 CloudStack 中扮演者很重要的角色，二级存储虚拟用来上传下载模板，上传下载镜像，在第一次创建虚拟机时对应的虚拟机节点还会从二级存储区拷贝模板或者镜像。而这些功能都是需要二级存储虚拟机的参与才能正常完成的。

管理节点通过二级存储虚拟机来管理模板、ISO、快照、卷这些资源当二级存储虚拟机发生问题时势必会影响到管理节点和二级存储模块的功能。为了评测系统虚拟机故障时，整个系统的容错能力，我们定义了如表 4-6 所示的故障集。

表 4-7 系统虚拟机故障模型

故障名称	故障描述	故障参数	预期效果
控制台虚拟机故障	控制台虚拟机宕机	ip 地址	网页控制台失效
二级存储虚拟机故障	二级存储虚拟机宕机	ip 地址	二级存储失效
虚拟路由器故障	虚拟路由器宕机	ip 地址	无法分配 ip 地址

4.1.3.3 网络故障

与分布式计算平台的网络故障类似，针对 CloudStack 上存在各个虚拟机节点，管理节点，系统虚拟机，这些节点之间都有可能因为网络的故障而导致正常通信功能的失常，我们将系统运行过程中可能发生网络故障进行总结，故障集定义如表 4-8 所示。

表 4-8 CloudStack 网络故障模型

故障名称	故障描述	故障参数	预期效果
管理节点和虚拟机网络故障	管理节点和虚拟机网络故障出现网络故障	ip 地址	管理节点和对应的虚拟机通信受到影响
管理节点和二级存储虚拟机网络故障	管理节点和二级存储虚拟机网络故障	ip 地址	管理节点和相应的二级存储的通信受到影响
管理节点和虚拟路由器网络故障	管理节点和虚拟路由器网络故障	ip 地址	管理节点和虚拟路由器的通信受到影响

4.1.3.4 管理节点资源故障

管理节点运行着 MySQL 数据库服务，还需要发送各种各样的控制命令，并对各个子组件例如虚拟机的状态，计算节点的状态，系统虚拟机的状态进行监控，当计算资源受到限制是一定会对管理节点的功能产生影响，为了评测当管理节点资源出现瓶颈时，CloudStack 的反应，我们设计了如下的如表 4-9 所示的故障集：

表 4-9 管理节点资源故障模型

故障名称	故障描述	故障参数	预期效果
内存占用故障	内存占用过高	ip 地址	部分控制命令无法执行
CPU 占用故障	CPU 占用过高	ip 地址	未知

4.2 多层次故障注入工具的实现

4.2.1 分布式计算平台故障注入工具的实现

根据我们定义的故障模型，本节将实现对 Hadoop 和 Spark 兼容的故障注入模块。

4.2.1.1 核心进程故障注入模块的具体实现

我们使用 kprobes 探针技术来作为此处使用的技术，Kprobes[37]是 Linux 中

的一个的内核调试工具，在 2.6 内核以上的版本中自带，可以监听内核函数，它提供了一个强行进入内核例程并且收集信息的接口，使用 Kprobes 可以收集处理器寄存器和全局数据结构等调试信息，开发者甚至可以使用 Kprobes 来修改寄存器值和全局数据结构的值[38]。

它的基本原理是这样的，首先，用户指定一个探测点，然后用户把自己定义的函数关联到到该探测点，那么，当内核执行到该探测点的时候，相应的关联函数就会被执行，用户定义的函数执行完后，就会返回执行正常的函数流程。Kprobes 提供 3 中不同类型的探针的使用方式，这三种各自具有各自不同的特点和优势。这三种探针类型分别是 kprobe、jprobe 和 kretprobes。

通过 Kprobes 提供的编程模型，我们可以通过内核编程的方式，设置好内核函数的名称，然后在内核进入和跳出对应的内核函数时我们可以获取到这个函数的参数列表，然后我们就可以在其中加入自己的代码，而且可以修改参数，通过这种方式我们可以在内核函数中模拟一些故障。例如寄存器故障和 Xen 的一些系统超级调用的故障还有 KVM 的一些相关的系统调用的故障我们都可以通过这种探针技术实现。具体到不同的故障注入的具体情况，需要对不同的函数进行分析，对参数进行分析，对上下文进行分析。

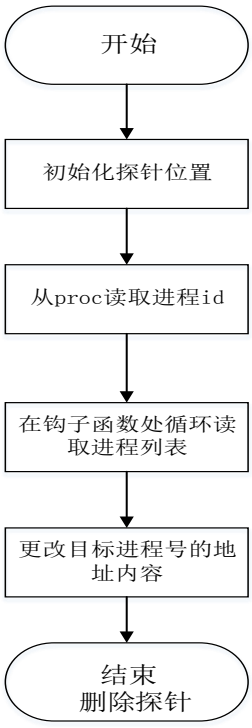


图 4-1 核心故障注入流程图

核心进程故障注入的程序流程图如图 4-1 所示。我们在 do_timer 处作为注册 kprobes 的探针位置，具体使用的探针类型是 kprobe,在每次进入 do_timer 之

前都会先进入我们自定义的函数中，我们在自定义的函数中从进程的根开始遍历进程列表，我们在用户态通过进程名获取到进程的进程 id，对于 java 进程我们使用 jps 命令可以获得进程名和进程 id，然后根据需要过滤即可，然后将此进程 id 通过 proc[39]传递给内核，当我们遍历进程列表时遍历到该进程后，就可以修改进程的寄存器，或者对应地址的内容，使进程发生异常。

4.2.1.2 网络通信故障模块的具体实现

我们使用 netfilter[40]提供的接口来实现在 Hadoop（Spark）层和 CloudStack 层注入网络故障，使用这种技术，我们可以在软件层次模拟出很多类型的网络故障。如图 4-2 所示，Netfilter 内核模块使用 hook 技术来对内核态数据包进行动态跟踪，预先定义了 5 个钩子函数，这 5 个钩子函数可以在数据包到达和离开时截获数据包然后进行相应的截获和修改。我们可以根据自己的需要，对数据包进行修改从而模拟出延迟、乱序，超时、丢包等网络故障。而且我们可以在其中加入 ip 过滤机制，只对目的 ip 或者源 ip 是我们所需的 ip 进行过滤，这样，我们在进行网络的故障注入时，就可以更加灵活，可以只阻断一个节点和某几个节点的通信，但不阻断我们对于这个节点的监听。

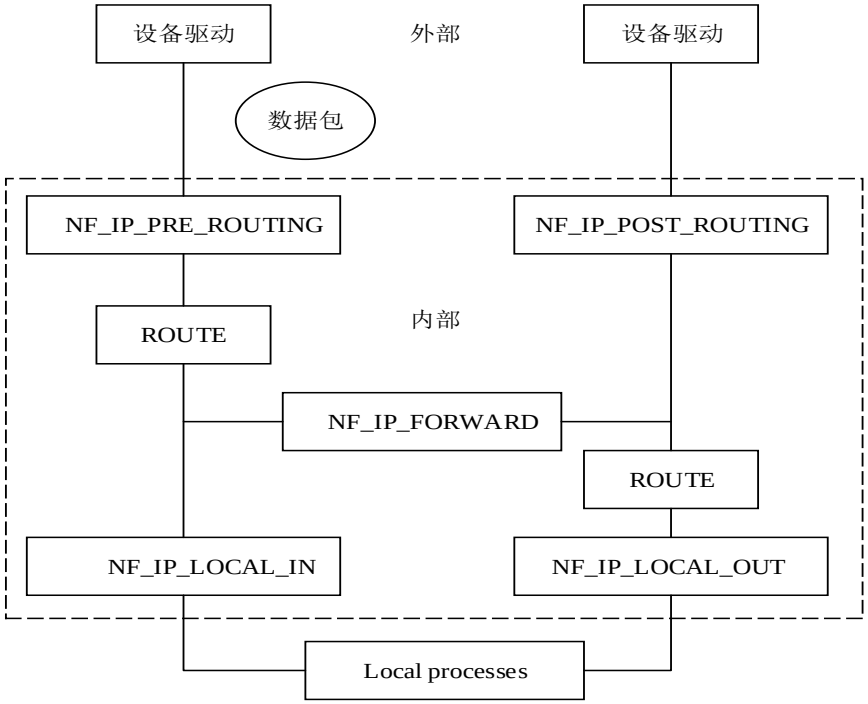


图 4-2 netfilter 架构图

通过这个技术，我们可以模拟很多生产环境中可能出现的网络故障，从而能够从网络的角度来评测云计算系统的健壮性。

我们在 NF_IP_PRE_ROUTING 这个钩子函数出进行处理，跳转到我们的钩

子函数，如果是网络延迟，我们就使用 mdelay 函数延迟指定的时间，然后返回 NF_ACCEPT 继续执行，如果是网络丢包，就按照一定的丢包率，随机的将数据包丢弃，不再处理。

为了进行更细致的网络的故障注入，我们还可以在其中加入 ip 的过滤，如果需要一个节点接收不到来自另外一个节点的数据包，但能正常接收别的节点的数据包，我们只需要在自己的钩子函数中加入 ip 的判断，如果是来自给定 ip 的数据包，我们选择接收，否则就进行丢弃操作。通过类似这样的方式，我们可以模拟出较丰富的网络故障状况。

4.2.1.3 资源占用故障模块的具体实现

内存的占用工具，我们首先通过 sysinfo 函数读取现在的内存的使用情况和总的内存量，然后根据需要达到的内存的占用百分比，计算出还需申请多少的内存，然后，通过 malloc 不断的申请内存，循环判断申请的内存量是否达到了我们需要的内存数量。

CPU 占用工具，我们首先通过 sysconf 函数读取现在的 CPU 的核心个数，然后根据我们需要达到的 CPU 的使用的阈值，让每个 CPU 都做相同的任务，我们根据 CPU 计算时间/空闲时间的比值来模拟 CPU 的占用率。

4.2.2 虚拟化层的故障注入工具的实现

4.2.2.1 软错误故障注入模块的具体实现

我们同样采用 kprobes 探针技术，在某个内核函数处设置探针，当内核运行到该函数时，我们通过 current->pid 来判断是否是目标进程号，如果是的话，我们通过传递过来的寄存器指针就可以实现对某个寄存器的更改。来达到一位翻转，交互，覆盖的目的。故障注入器会在目标进程例如 qemu-kvm 进程设置断点，当断点被触发的时候，会更改对应的目标。

4.2.2.2 客户操作系统错误行为故障注入模块的具体实现

同样采用内核模块的编程方法，故障注入模块作为内核模块安装在客户操作系统上，故障注入器可以识别正在运行在客户操作系统上的进程然后（1）随机的更改进程的状态，通过翻转数据段，代码段，寄存器的一位或者多位来实现。（2）抛出 CPU 异常：例如除以 0，无效的操作。

4.2.2.3 性能故障注入模块的具体实现

采用 kprobes 探针技术，使用探针绑定用户确定断点的地址（系统调用的名称），我们将 target_pid 通过 proc 文件系统传给内核，在进程运行到执行 ioctl 函数的时候，会跳到我们定义的函数，我们在函数中通过 current->pid 获取当

前的进程号，通过该进程号与目标进程号对比来判断是否是我们需要注入故障的目标进程，如果是的话，执行我们的注入故障的逻辑，使线程延迟给定的时间，如果不是的话，函数返回，执行正常的函数逻辑，需要注意的是 sleep 在具体实现时需要使用 Linux 的 mdelay 函数而不能使用 msleep 函数，因为 kprobes 是运行在中断上下文中的，中断处理的时候，不应该发生进程的切换，因为在中断上下文中，唯一能打断当前中的只有更高优先级的中断，它不会被进程打断，如果在中断上下文中休眠，则没有办法唤醒它。通过这种方式，我们可以使多个 qemu-kvm 进程发生在调用 ioctl 时产生延迟这种情景。进而模拟出对资源的竞争的情形。

```
int (*my_ioctl) (struct inode ...)
{
    if(current->pid == target_pid)
        sleep(delay time)

    jprobe_return();
}
```

图 4-3 性能故障代码

4.2.2.4 维护故障注入模块的具体实现

当前版本的 Linux 实现中支持 CPU 的热插拔，也就说有动态的打开或关闭 CPU 核心的能力。一个使用者可以通过更该/sys/devices/system/cpu/cpuX/online 的值来实现改变 cpu 的状态，这个会调用内核函数来更改 CPU 的使用状态。

故障注入器通过通过调用相同的内核函数来实现更改目标 CPU 的状态。用户首先设置断点的位置，当断点到达时，故障注入器可以通过调用内核函数来到达更改目标 CPU 的状态这个目的。或者注入故障器也可以通过直接修改 /sys/devices/system/cpu/cpuX/online 来达到动态修改 CPU 的核心数的目的。通过 echo 0 > /sys/devices/system/cpu/cpu1/online 这条命令我们就可以使 cpu1 关闭，echo 1 > /sys/devices/system/cpu/cpu1/online 可以使对应的 cpu1 关闭。

4.2.3 云平台管理软件栈的故障注入工具的实现

4.2.3.1 存储故障注入模块的具体实现

我们采用修改 VFS 跳转表中对应的操作函数的方式来实现存储故障工具。VFS 是一种位于具体的文件系统之上的抽象层次。如**错误!未找到引用源。**4 所示，本文提出了通过 VFS 文件系统层[41]来模拟上层文件读写故障的方法。我们根据需要注入的文件的命名或者文件目录，首先打开对应的文件描述符 filep，

然后通过 filep->f_dentry->d_inode->i_fop 获取到文件操作函数的指针，将旧的保存，将指针指向自己的读或者写的函数，注意函数名的定义要和旧的一样。这样就可以实现对对应的文件目录或者文件名实现读写的故障，如图 4-5 所示。

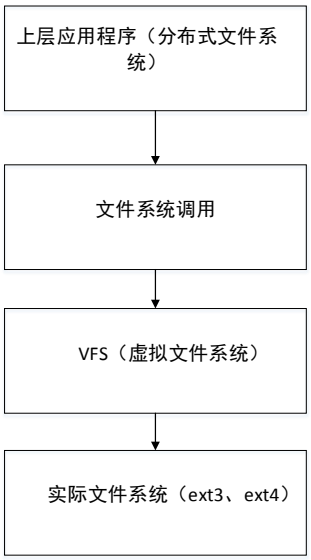


图 4-4 VFS 故障注入层次原理图

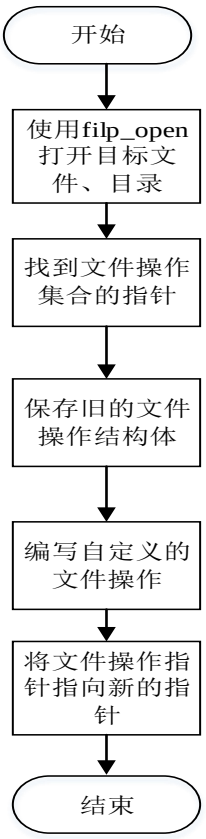


图 4-5 文件系统故障流程图

4.2.3.2 系统虚拟机故障注入模块的实现

我们仍然采用 kprobes 探针的方式，在钩子函数处遍历所有的进程列表，然后修改关键寄存器的值，使得系统虚拟机宕机。和关键进程故障实现流程类似。

4.2.3.3 网络故障注入模块的实现

和 4.4.4 节中网络故障的实现原理类似，不同之处是网络故障的位置不同。

4.2.3.4 资源占用故障注入模块的实现

和 4.4.4 节中的资源占用故障实现原理类似，不同之处是资源占用故障的位置不同。

4.3 本章小结

本章从故障注入的角度出发，通过前面对各个层次各个组件的架构和功能的分析，分别设计了针对分布式计算平台的的故障注入工具，针对虚拟化层的故障注入工具，针对云平台管理栈的故障注入工具。针对分布式计算平台的故障工具包含进程级的故障注入模块，功能级的故障模块，网络故障注入模块，资源占用故障注入模块。针对虚拟化层设计了软错误故障注入模块，维护故障注入模块，性能故障注入模块。针对云平台管理栈设计了存储故障注入模块，网络故障注入模块，系统虚拟机故障注入模块，管理节点资源故障注入模块。

第5章 故障注入评测实验

5.1 实验环境的搭建

为了进行云计算平台的故障注入，我们搭建了小型的云计算平台。我们采用超云服务器作为硬件服务器，采用 XenServer 和 KVM 作为虚拟机管理器来进行硬件的虚拟化。在 XenServer 和 KVM 上创建 Centos 系统作为客户虚拟机，在 Centos 上部署 Hadoop 和 Spark 平台来运行各种 Hadoop 和 Spark 的应用程序。此外我们还选择了 CloudStack 作为云平台的管理软件栈，来综合管理所有的硬件资源，虚拟机资源。

测试端为 Windows 系统，安装有对应的 JDK 开发包和运行环境。逻辑上的部署如图 5-1 所示。

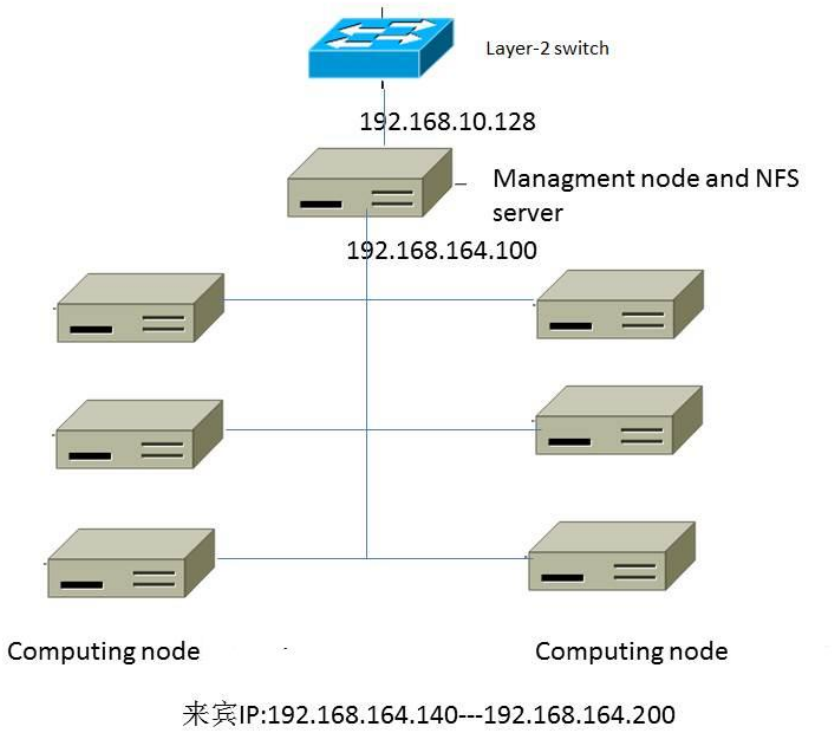


图 5-1 实验部署图

我们选用超云服务器作为我们的实验环境使用的物理机。超云服务器物理机 配置如表 5-1 所示。实验使用的 XenServer 版本为 6.1，KVM 版本为 centos6.3 对应的版本。CloudStack 版本为 4.3。虚拟机的操作系统为 Centos 6.3。

表 5-1 超云服务器硬件配置表

机器名称	CPU	内存	硬盘	网卡
超云服务器	24*Intel® Xeon® CPU E5-2620 @ 2.10GHz	64GB	1.81T	千兆

5.2 多层次故障注入实验

5.2.1 Hadoop 和 Spark 故障注入测试对比

5.2.1.1 测试步骤

首先运行 Hadoop 或者 Spark 负载。 Hadoop 的使用自带的 terasort 基准测试程序，首先生成 500M 的数据，然后对这 500M 的数据进行排序。Spark 使用计算 Pi 的负载。通过 ip 连接要注入故障的节点。

然后根据具体的故障类型，选择故障和故障参数。如果是进程故障要配置好进程名称，如果是网络故障要选择网卡和网络故障的参数例如网路延迟 100ms，如果是内存占用故障，要输入相应占用率，例如 80%。

最后收集负载的运行情况和系统的反应。然后进行分析对比。

5.2.1.2 核心进程的故障注入实验

我们分别对 Hadoop 和 Spark 注入核心进程故障，并观察它们在遇到核心进程故障后的反应。下面分别是 Hadoop 和 Spark 在遇到进程故障时各自的反应。

图 5-2 为向 Hadoop 平台注入 Map 进程故障后 Hadoop 平台的反应，系统检测到该任务的失败，然后进行重做。

```
15/11/22 16:31:44 INFO mapred.JobClient:  map 100% reduce 0%
15/11/22 16:31:45 INFO mapred.JobClient: Task Id : attempt_201602291555_0003_r_000000_0, Status : F
AILED
java.lang.Throwable: Child Error
        at org.apache.hadoop.mapred.TaskRunner.run(TaskRunner.java:271)
Caused by: java.io.IOException: Task process exit with nonzero status of 143.
        at org.apache.hadoop.mapred.TaskRunner.run(TaskRunner.java:258)
15/11/22 16:31:55 INFO mapred.JobClient:  map 100% reduce 33%
15/11/22 16:31:56 INFO mapred.JobClient:  map 100% reduce 100%
15/11/22 16:31:57 INFO mapred.JobClient: Job complete: job_201602291555_0003
15/11/22 16:31:57 INFO mapred.JobClient: Counters: 30
```

图 5-2 map 进程故障图

从图 5-3 为向 Spark 平台注入 CoarseGrainedExecutorBackend 进程故障后，系统的反应。我们可以看到当注入 CoarseGrainedExecutorBackend 进程故障时，对应的 Executor 丢失，这是因为 CoarseGrainedExecutorBackend 进程是 Executor 的守护进程，它的崩溃导致相应的 Executor 也崩溃，进而导致这个 Executor 分

配的任务丢失，相应的容错策略是注册了新的 Executor，然后重新执行丢失的任务。

通过各个实验我们可以发现在非核心进程遇到故障时，Hadoop 和 Spark 都可以通过重做机制来使得出错的任务重新执行。在没有使用 ZooKeeper 时，如果 Hadoop 和 Spark 的关键进程比如 Hadoop 的 NameNode 进程和 JobTracker 进程，Spark 的 Master 进程遇到异常退出时或者没有反应时，整个系统都会崩溃。当我们通过部署 ZooKeeper 后，ZooKeeper 可以通过自己的功能来帮助系统来同时管理多个备份的关键进程，备份的处于 stand-by 状态，有一个处于正常提供功能的状态。当提供服务的出现问题时，ZooKeeper 会选取新的节点来代替失效的节点来提供服务这样提高了整个平台的可用性。

```
16/06/08 11:57:47 ERROR scheduler.TaskSchedulerImpl: Lost executor 0 on 192.168.16.145: remote Rpc client
disassociated
16/06/08 11:57:47 INFO scheduler.TaskSetManager: Re-queueing tasks for 0 from TaskSet 0.0
16/06/08 11:57:47 WARN scheduler.TaskSetManager: Lost task 399.0 in stage 0.0 (TID 403, 192.168.16.145): E
xecutorLostFailure (executor 0 lost)
16/06/08 11:57:47 WARN scheduler.TaskSetManager: Lost task 398.0 in stage 0.0 (TID 402, 192.168.16.145): E
xecutorLostFailure (executor 0 lost)
16/06/08 11:57:47 WARN remote.ReliableDeliverySupervisor: Association with remote system [akka.tcp://spark
Executor@192.168.16.145:54233] has failed, address is now gated for [5000] ms. Reason: [Disassociated]
16/06/08 11:57:47 WARN scheduler.TaskSetManager: Lost task 397.0 in stage 0.0 (TID 401, 192.168.16.145): E
xecutorLostFailure (executor 0 lost)
16/06/08 11:57:47 WARN scheduler.TaskSetManager: Lost task 400.0 in stage 0.0 (TID 404, 192.168.16.145): E
xecutorLostFailure (executor 0 lost)
16/06/08 11:57:47 INFO scheduler.DAGScheduler: Executor lost: 0 (epoch 1)
16/06/08 11:57:47 INFO storage.BlockManagerMasterEndpoint: Trying to remove executor 0 from BlockManagerMa
ster.
16/06/08 11:57:47 INFO storage.BlockManagerMasterEndpoint: Removing block manager BlockManagerId(0, 192.16
8.16.145, 33629)
16/06/08 11:57:47 INFO storage.BlockManagerMaster: Removed 0 successfully in removeExecutor
16/06/08 11:57:47 INFO client.AppClient$ClientEndpoint: Executor updated: app-20160608115731-0000/0 is now
 EXITED (Command exited with code 143)
16/06/08 11:57:47 INFO cluster.SparkDeploySchedulerBackend: Executor app-20160608115731-0000/0 removed: Co
mmand exited with code 143
16/06/08 11:57:47 INFO cluster.SparkDeploySchedulerBackend: Asked to remove non-existent executor 0
16/06/08 11:57:47 INFO client.AppClient$ClientEndpoint: Executor added: app-20160608115731-0000/3 on worke
r-20160608115535-192.168.16.145-49276 (192.168.16.145:49276) with 4 cores
16/06/08 11:57:47 INFO cluster.SparkDeploySchedulerBackend: Granted executor ID app-20160608115731-0000/3
on hostPort 192.168.16.145:49276 with 4 cores, 1024.0 MB RAM
16/06/08 11:57:47 INFO client.AppClient$ClientEndpoint: Executor updated: app-20160608115731-0000/3 is now
 RUNNING
16/06/08 11:57:47 INFO client.AppClient$ClientEndpoint: Executor updated: app-20160608115731-0000/3 is now
 LOADING
16/06/08 11:57:47 INFO scheduler.TaskSetManager: Starting task 400.1 in stage 0.0 (TID 417, 192.168.16.105
, PROCESS_LOCAL, 2164 bytes)
```

图 5-3 CoarseGrainedExecutorBackend 进程故障图

核心进程故障注入实验表明，Hadoop 和 Spark 都能容忍大部分的进程错误，对此类的故障有较好的容错能力。

5.2.1.3 网络故障注入测试

我们对 Hadoop 和 Spark 系统分别注入网络报文延迟故障、网络报文丢包故障，并运行相应的负载，观察网络故障对系统本身和负载的执行情况的影响。以此来测试它们对网络故障的容错能力。

（1） 网络报文延迟故障测试

如表 5-2 所示，随着网络延迟的增加，任务的执行时间缓慢增加，当网络延迟为 2 秒时，Hadoop 会出现心跳故障，无法正常通信，任务的执行实现未知。而 Spark 在 网络延迟为 2 秒时任务的执行时间与故障注入前相比增加了 20 倍。如图 5-4 所示，网络故障导致 Hadoop 出现 IPC（进程间）通信错误，任务无法正常进行。将网络延迟去掉后，重新执行任务，任务正常运行没有出现由于异常而暂停的现象。

表 5-2 延迟测试结果

网络延迟时间	故障注入前	20ms	200ms	600ms	2s
Hadoop 执行时间	96s	99s	120s	184s	未知
Spark 执行时间	21s	23s	46s	126s	408s

图 5-4 Hadoop 网络延迟故障图

图 5-5 是 Spark 在网络延迟为 2 秒时系统发生的故障，从系统的打印信息我们可以看到，master 节点发生读超时，task 4 任务在阶段 0.0 不能正常读取。需要重新执行 task4 任务。这使得任务的运行时间由于通信的问题急剧增加。

我们可以发现，Hadoop 和 Spark 对网络延迟故障都有一定程度的容错能力，当网络故障延迟故障不严重时，只会轻微的影响到进程和任务间的通信，当网络故障较为严重时，会使得任务失败，然后重做任务，大量增加任务的执行时

间。

```
16/06/01 20:06:09 INFO scheduler.TaskSetManager: Finished task 998.0 in stage 0.
0 (TID 998) in 102 ms on 192.168.16.105 (996/1000)
16/06/01 20:07:38 WARN scheduler.TaskSetManager: Lost task 4.0 in stage 0.0 (TID
 4, 192.168.16.145): java.net.SocketTimeoutException: Read timed out
        at java.net.SocketInputStream.socketRead0(Native Method)
        at java.net.SocketInputStream.read(SocketInputStream.java:152)
        at java.net.SocketInputStream.read(SocketInputStream.java:122)
        at java.io.BufferedInputStream.read1(BufferedInputStream.java:273)
        at java.io.BufferedInputStream.read(BufferedInputStream.java:334)
        at sun.net.www.MeteredStream.read(MeteredStream.java:134)
        at java.io.FilterInputStream.read(FilterInputStream.java:133)
        at sun.net.www.protocol.http.HttpURLConnection$HttpInputStream.read(Http
JRLConnection.java:3066)
        at sun.net.www.protocol.http.HttpURLConnection$HttpInputStream.read(Http
JRLConnection.java:3060)
        at org.apache.spark.util.Utils$$anonfun$copyStream$1.apply$mcJ$sp(Utils.
scala:284)
        at org.apache.spark.util.Utils$$anonfun$copyStream$1.apply(Utils.scala:2
```

图 5-5 Spark 网络延迟故障图

（2）网络报文丢包故障注入测试

表 5-3 网络丢包故障测试结果

网络丢包率	故障注入前	10%	30%	60%
Hadoop 执行时间	96s	106s	未知	未知
Spark 执行时间	21s	23s	26min	未知

当一个节点的网络丢包率的故障为 30%时，Spark 和 Hadoop 都出现了严重的故障，通过实验我们可以发现 Hadoop 和 Spark 对网络故障都有一定的容忍限度，在 Hadoop 和 Spark 的配置文件中都有对用配置可以配置心跳的阈值，当网络故障超过使得心跳超过阈值时，Hadoop 或者 Spark 就会将对应的节点或者进程视为有问题的节点，当网络故障不明显的时候，只是轻微的影响任务的执行时间，当网络故障很严重的时候，会引起任务失败、节点被标记为问题节点，这会严重影响系统的性能。

5.2.1.4 资源占用故障注入测试

在内存占用率故障方面，没有注入故障时，任务的执行时间为 1 分 36 秒即 96 秒。在一个 slave 节点注入内存占用 80%时的执行时间为 109 秒。一个 slave 节点内存占用为 95%时的执行时间为 129 秒

当内存的或者 CPU 的占用达到较高的水平时，会出现和网络故障类似的现象，某些 RPC 通信无法正常运行，心跳通信失败。如下图所示。

```
2016-06-02 02:28:05,392 INFO org.apache.hadoop.mapred.JobInProgress: Job job_201606020120
_0007 initialized successfully with 8 map tasks and 1 reduce tasks.
2016-06-02 02:28:05,405 INFO org.apache.hadoop.mapred.JobTracker: Adding task (JOB_SETUP)
 'attempt_201606020120_0007_m_000009_0' to tip task_201606020120_0007_m_000009, for track
er 'tracker_slave2:localhost/127.0.0.1:39304'
2016-06-02 02:28:07,539 ERROR org.apache.hadoop.security.UserGroupInformation: Priviledge
dActionException as:hadoop cause:java.io.IOException: java.lang.NullPointerException
2016-06-02 02:28:07,539 INFO org.apache.hadoop.ipc.Server: IPC Server handler 8 on 9001,
call heartbeat(org.apache.hadoop.mapred.TaskTrackerStatus@2e0b5c56, false, false, true, 1
2914) from 192.168.10.44:40104: error: java.io.IOException: java.lang.NullPointerExceptio
n
java.io.IOException: java.lang.NullPointerException
        at org.apache.hadoop.mapred.JobInProgress.updateTaskStatus(JobInProgress.java:125
7)
        at org.apache.hadoop.mapred.JobTracker.updateTaskStatuses(JobTracker.java:4943)
        at org.apache.hadoop.mapred.JobTracker.processHeartbeat(JobTracker.java:3838)
        at org.apache.hadoop.mapred.JobTracker.heartbeat(JobTracker.java:3532)
        at sun.reflect.GeneratedMethodAccessor9.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.j
ava:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at org.apache.hadoop.ipc.RPC$Server.call(RPC.java:578)
        at org.apache.hadoop.ipc.Server$Handler$1.run(Server.java:1393)
```

图 5-6 Hadoop 内存占用故障图

CPU 占用率故障注入实验，没有注入故障时时间为 96 秒，注入使用率为 80%时，相同的任务的执行时间为 105 秒，注入使用 90%时有时会发生故障，任务的执行时间为 140 秒。

Spark 的使用自带的 Pi 来作为执行的任务，没有注入故障的时候时间为 22 秒，CPU 使用率为 80%时执行时间 23 秒，CPU 使用率为 90%时任务的执行时间为 26 秒。

5.2.2 Xen 和 KVM 故障注入测试对比

虚拟机故障注入实验的负载我们选用 ApacheBench，这是一个 HTTP 服务器的评测工具，它可以给运行在虚拟机上的 Apache 服务器集中的发送 HTTP 请求。我们在测试时首先运行 ApacheBench 负载，然后通过我们的用户交互接口注入定义的各种虚拟机故障，然后观察虚拟机管理器和虚拟机的反应。

对于 KVM 虚拟机化系统，我们对它的客户操作系统，qemu-kvm，KVM 内核模块，宿主机的 Linux 内核这几个组件都进行了故障的注入实验。对于 Xen 虚拟化系统，我们对客户操作系统，qemu-dm 和 Xenstored，Dom0 的 Linux 内核，Xen hypervisor 均进行了故障注入的实验。我们选择 qemu-dm 和 Xenstored 的原因是它们由 Xen 在用户级提供的可以用来创建和控制客户操作系统的组件，它们的故障很有可能引起 Xen 虚拟化系统的故障。

5.2.2.1 软错误测试

在 XenServer 和在 KVM 这两种虚拟机管理器下，在修改非关键寄存器时系

统都不会发生故障，但是在修改 ip 寄存器，段寄存器时都会出现故障，修改客户虚拟机的 ip 寄存器会使得这个虚拟机发生崩溃。修改宿主机的 ip 寄存器会使得宿主机崩溃。如图 5-10 所示就是注入 ip 寄存器一位翻转故障后的系统的反应，系统发生段错误，然后崩溃。

```
init[1]: segfault at 7fd1a02af6af ip 00007fd1904ffebd sp 00007fffb17134f0 error
6 in libc-2.12.so[7fd190455000+18a000]
Kernel panic - not syncing: Attempted to kill init!
Pid: 1, comm: init Not tainted 2.6.32-279.el6.x86_64 #1
Call Trace:
 [<ffffffff814fd11a>] ? panic+0xa0/0x168
 [<ffffffff813140a6>] ? get_current_tty+0x66/0x70
 [<ffffffff81070bd2>] ? do_exit+0x862/0x870
 [<ffffffff81081a5d>] ? __sigqueue_free+0x3d/0x50
 [<ffffffff81070c38>] ? do_group_exit+0x58/0xd0
 [<ffffffff81085866>] ? get_signal_to_deliver+0x1f6/0x460
 [<ffffffff8100a2d5>] ? do_signal+0x75/0x800
 [<ffffffff81060a83>] ? wake_up_new_task+0xd3/0x120
 [<ffffffff814fd223>] ? printk+0x41/0x46
 [<ffffffff8100aaf0>] ? do_notify_resume+0x90/0xc0
 [<ffffffff8100bb5c>] ? retint_signal+0x48/0x8c
```

图 5-10 系统宕机

当基于 libvirt 的管理模块挂起的时候，qemu-kvm 进程会变为僵尸进程。我们向 qemu-kvm 进程注入了随机了寄存器翻转故障，这个寄存器级的错误导致了 qemu-kvm 进程成为了僵尸进程（进程状态变为无效状态），当我们查询对应的虚拟机的状态时,管理系统会挂起。这个虚拟机对应的是无效的 qemu-kvm 状态。

5.2.2.2 客户操作系统的错误行为测试

我们经过测试发现，在 KVM 中，错误不会从客户操作系统传递到宿主操作系统。而在 Xen 中，大部分情况下错误也会被隔离在客户操作系统内，只有在某些情况下，例如迁移故障，错误会从客户操作系统传递到宿主操作系统，宿主操作系统表现为宕机。

5.2.2.3 性能故障测试

我们对一个正在运行在 KVM 管理器上的虚拟机的某个虚拟 CPU 注入性能故障，查看对应的 qemu-kvm 的进程树，我们发现，当我们往某个虚拟 CPU 线程的所有 sys_ioctl 函数注入延迟 20ms 的故障后，系统反应变得很慢，当我们加大延迟到 20s 后，宿主机的反应延迟也很大，接近死机状态。

如果我们只让每个虚拟 CPU 的线程在执行 sys_ioctl 时有一次延迟，加入延迟的 CPU 的数量从 1 到 4 变化，我们没有发现内核崩溃的情况。

当我们对其中的一个虚拟 CPU 线程注入异常结束的故障后，对应的虚拟机无法控制，虚拟机异常关闭。虚拟机管理器丢失与虚拟机的连接。

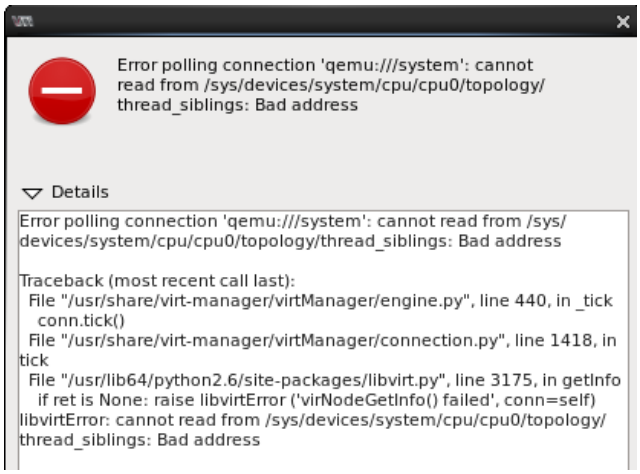


图 5-11 丢失和虚拟机的连接

5.2.2.4 维护故障测试

在 KVM 中，如果将 CPU 的核心打开或者关闭，宿主机或者客户操作系统都不会出现异常。在实验中，我们观察到，当打开或者关闭一个物理 CPU 核心的时候，虚拟机管理器不会影响到正在运行的虚拟机的状态和别的服务的状态。在 Xenserver 中的宿主机和客户操作系统中打开和关闭 CPU 核心的操作也是都正常运行的。操作之前有 4 个 CPU 核心，关闭其中两个后只有 2 个核心。但系统没有异常的反应。表 5-4 为虚拟化层的故障注入的结果对比。

表 5-4 虚拟化层故障结果对比

故障类型	Hypervisor	故障位置	虚拟机管理器反应
维护故障	KVM	物理机上的 CPU 核心	正常
		客户机的 CPU 核心	正常
	Xen	物理机上的 CPU 核心	正常
		客户机的 CPU 核心	正常
寄存器故障	KVM	ax,bx 等非关键寄存器	正常
		ip 关键寄存器	系统崩溃
	Xen	ax,bx 等非关键寄存器	正常
		ip 关键寄存器	系统崩溃
性能故障	KVM	qemu-kvm 线程	系统崩溃
	Xen	N/A	N/A
客户虚拟机错误行为	KVM	虚拟机各个寄存器	正常
	Xen	虚拟机各个寄存器	部分正常

5.2.3 CloudStack 故障注入测试

5.2.3.1 存储故障注入测试

向二级存储器注入写故障，然后我们通过 web 接口注册镜像，我们发现，无法正常上传镜像，系统报错。该服务失效。如图 5-12 所示，由于存储主机的响应超时一直处于未就绪状态。

向主存储器添加读故障，本地的读写无法正常运行。原因是 CloudStack 在存储上并没有相应的容错策略，没有类似 hdfs 的备份策略。因此当一份数据无法读取时，就会出现故障。

通过对存储故障的注入测试我们发现，CloudStack 在存储层面没有类似多机备份等容错策略，这是今后需要改进的一个方面。

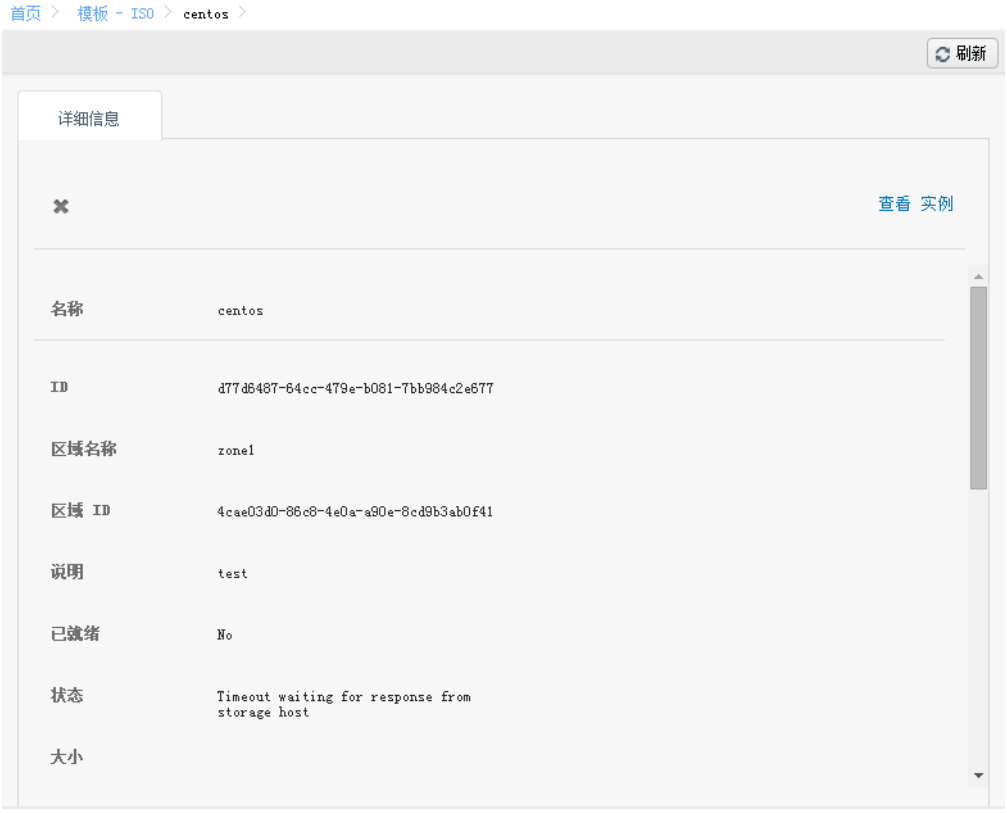


图 5-12 二级存储响应超时

5.2.3.2 系统虚拟机故障注入测试

二级存储虚拟机故障，向二级存储虚拟机注入核心进程故障，使该虚拟机出现宕机，二级存储器崩溃，与二级存储器相关的任务均无法正常运行。比如注册模板，注册 ISO 等。

控制台虚拟机故障，向控制台虚拟机注入核心进程比如 1 号进程故障，使得该虚拟机出现宕机之后无法从浏览器端登录虚拟机的控制台，该功能受到影

响。

虚拟路由器出现问题后会对系统提供产生很大的影响，我们发现，当创建很多的虚拟机后，会出现由于资源不足而产生很严重的故障，导致通过 CloudStack 的 web ui 无法打开，创建虚拟机，经过查找和研究，发现当把虚拟路由器销毁后，生成新的虚拟路由器，才使得整个管理系统恢复正常，可以正常的启动和创建虚拟机。与存储故障不同，CloudStack 对系统虚拟机的控制方面自带有容错策略，当系统虚拟机崩溃后，CloudStack 会检测到这种情况，并且在一定时间后重启相应的系统虚拟机来对这种故障进行处理。

5.2.3.3 网络故障注入测试

向一个计算节点上的虚拟机注入网络故障，注入延迟 1s 的故障，然后在 CloudStack 的控制界面关闭添加网络故障的虚拟机，如图 5-13 和图 5-14 所示，发现该虚拟机长时间卡在了 stopping 状态，很长时间后才正常关闭。通过这个测试用例我们可以发现网络的延迟无论对于什么系统都是很重要的一个环节，因为所有的通信都是通过网络来进行的。

图 5-13 虚拟机状态图

图 5-14 虚拟机状态图

在向管理节点和计算节点注入网络故障后，如果网络故障较为严重则会受到较大的影响。比如如果网络的延迟较大导致管理节点无法和计算节点正常通信，管理节点会认为该节点出现故障，将该节点丢弃。如果启用了主机的高可用性，并且主存储是网络存储（NFS）的话，该主机上的虚拟机会被自动的迁移到其余的可用的计算节点上。

5.2.3.4 管理节点资源故障注入测试

当内存的占用率正常时，创建虚拟机流程正常，逐步增加内存占用率，直到内存占用率达到 95%左右后,如果再创建虚拟机就会出现由于资源不足而产生故障。如图 5-15 所示。

图 5-15 无法启动虚拟机

在日志中出现如图 5-16 所示的错误，从日志中可以看到由于资源不足导致无法启动虚拟机。而且此时将内存占用故障取消后，仍然无法创建新的虚拟机，所以猜测 CloudStack 的这个部分有 bug 存在，估计是在内存资源不够后，某些状态没有及时更新，一直记录为资源不够，进而导致此故障的出现对于此故障没有进行很好的处理希望能在今后的版本中改进。经过我们测试只有当重启虚拟路由器后启动虚拟机的功能才恢复正常，具体原因还有待进一步研究。

```
2016-06-02 16:00:34,692 WARN  [o.a.c.alerts] (Job-Executor-19:ctx-6c67fc68 ctx-551d6518)  alertType:: 8 //
 dataCenterId:: 1 // podId:: 1 // clusterId:: null // message:: Failed to deploy Vm with Id: 106, on Host
with Id: 4
2016-06-02 16:00:34,796 ERROR [c.c.a.ApiAsyncJobDispatcher] (Job-Executor-19:ctx-6c67fc68) Unexpected exce
ption while executing org.apache.cloudstack.api.command.user.vm.DeployVMCmd
com.cloud.utils.exception.CloudRuntimeException: Unable to start a VM due to insufficient capacity
        at com.cloud.vm.VirtualMachineManagerImpl.start(VirtualMachineManagerImpl.java:865)
        at org.apache.cloudstack.engine.cloud.entity.api.VMEntityManagerImpl.deployVirtualMachine(VMEntity
ManagerImpl.java:237)
        at org.apache.cloudstack.engine.cloud.entity.api.VirtualMachineEntityImpl.deploy(VirtualMachineEnt
ityImpl.java:207)
        at com.cloud.vm.UserVmManagerImpl.startVirtualMachine(UserVmManagerImpl.java:3564)
        at com.cloud.vm.UserVmManagerImpl.startVirtualMachine(UserVmManagerImpl.java:3171)
        at com.cloud.vm.UserVmManagerImpl.startVirtualMachine(UserVmManagerImpl.java:3157)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:622)
        at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)
        at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodIn
vocation.java:183)
        at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation
.java:150)
```

图 5-16 日志异常

5.2.4 综合故障注入实验

本文前面已经使用设计的 3 个层次的故障注入注入工具对云平台的 3 个层次分别进行了故障注入的实验。然而，本平台最大的优点应该是可以同时对多个层次的故障进行综合的注入，而且是可定时可批量注入的。下面我们将会同时注入多个层次的故障来验证本平台的功能同时测试云系统的健壮性。

下面我们首先考虑不同层次见故障的影响和传播。在一个层次上注入故障，故障对另一层次上负载或者系统的影响。

我们在虚拟机层注入 CPU 关闭几个核心的故障，在 Spark 层次运行排序的负载，我们观测这种情况会有什么发生。我们运行的 Spark 的负载是计算 Pi 的运算任务。我们分别在两个 Worker 节点上关闭 3 个 CPU 核心，对任务的执行时间没有明显的影响，更改前时间为 21 秒，更改后仍未 21 秒左右，该故障对任务的影响很小，原因是该负载对 CPU 的利用率不是很高，只有 15%左右，不是计算密集型任务，所以在减少 CPU 核心时没有明显影响。而且我们在执行任务的过程中打开关闭核心对任务的执行竟然没有影响这让我们很意外。

还比如，我们在执行一个 Spark 任务时，一个虚拟机由于某种原因意外关闭，这会导致所有在该节点上执行的任务的全部丢失。这相当于发生了严重的网络故障，Spark 主节点与该虚拟机节点的所有通信全部中断。在该节点上执行的任务需要按照记录的执行的动作全部执行。按照 Spark 自身的容错机制进行恢复。

5.3 本章小结

本章使用第三章设计的故障注入平台和第四章设计的多层次的故障工具对分布式计算平台的容错能力，虚拟化平台的容错能力，云管理栈的容错能力进行了故障注入的测试。实验结果表明，云计算系统的各个层次都具有较强的容错能力。但是对于某些情况还不具备容错能力。

结　论

随着云计算技术的发展，越来越多的云计算平台也出现在了人们的视野中，各种各样的云计算平台有着各自不同的特性，提供了丰富的管理功能，我们从云计算平台的容错能力的角度出发，设计提出了一套较为通用的多层次的云计算故障注入平台，该平台可以针对云计算的不同层次进行故障故障，并能够通过图形界面或者脚本实现较为自动化的故障的注入和收集，而且能对系统做到一定程度的资源监控。各个故障注入工具以模块的方式添加和删除，方便了本平台的扩展。我们本身集成了分布式计算平台故障注入工具模块，虚拟化故障注入工具模块，云平台管理栈故障注入模块。在这三个层面我们分别进行了故障注入的测试。

在分布式计算平台层面，我们通过对比 Spark 和 Hadoop 这两种典型的系统，设计了针对核心进程的故障注入测试工具，针对网络的故障注入工具，针对资源占用的故障注入测试。通过实验我们发现，Hadoop 和 Spark 均可以对非核心进程故障有较好的容错能力，在部署了 ZooKeeper 后，对单点故障也可以有很好的容错能力。在针对网络的故障注入测试中，我们发现 Hadoop 和 Spark 对于较小的网络延迟等故障在一定的阈值范围内可以忍受，但会稍微影响任务的运行时间，当网络故障较大时，例如有较大的丢包率 30%时，任务的执行时间受到了很大的影响，有些任务的心跳丢失，通信的代价急剧提高。资源占用的故障也会在不同程度上影响任务的执行时间。

在虚拟化层面，我们对比了 KVM 和 Xen 这两种虚拟化技术的容错能力，我们发现无论是对于 KVM 还是对于 Xen 客户虚拟机的故障都不会传递给宿主机。关键系统调用的故障的故障会对虚拟机产生很严重的故障，例如 Xen 的一些关键的超级调用出现故障时，系统会出现宕机等表现。

在云平台管理栈层面，我们对 CloudStack 设计了故障注入工具，并对它的存储故障，系统虚拟机故障，管理节点资源故障，网络故障进行了故障注入的测试。实验结果表明，存储故障会严重影响相应的存储功能的提供，而系统虚拟机故障通过 CloudStack 自带的自动检测与重启系统虚拟机的机制可以很好的容忍这种故障。管理节点的资源故障会影响到控制命令的发出，严重时无法创建启动虚拟机。网络故障会使得管理功能受到很大的影响。

对于后续的研究工作，本文对 Hadoop 和 Spark 的故障模块的设计还是基于较粗粒度的故障注入，后续可以在更细的粒度上进行。对于 KVM 和 Xen 的研究对比的也不是很充分，这也是可以研究的方向。而在云平台管理栈方面，本

文只对 CloudStack 进行了故障的分析，后续可以对 OpenStack，Eucalyptus 等平台进行故障的注入和健壮性的对比分析。在实验上，还缺乏较大规模的故障注入的实验。

参考文献

[1] 刘缙, 朱家稷, 张海勇. 大规模云计算平台的技术挑战[J]. 程序员, 2012 (2): I0007-I0009.

[2] 石山. 军事信息云的网络架构设计及作战指挥辅助决策研究[D]. 西安电子科技大学, 2012.

[3] 张辉. 基于 Xen 的虚拟机快照技术研究[D]. 东南大学,2012.

[4] 潘慧, 朱信忠, 赵建民, 等. 基于 Hadoop 云测试体系架构的设计[J]. 计算机工程与科学, 2013, 35(10): 72-78.

[5] Kumar R, Jain K, Maharwal H, et al. Apache CloudStack: Open source infrastructure as a service cloud computing platform[J]. Proceedings of the International Journal of advancement in Engineering technology, Management and Applied Science, 2014: 111-116.

[6] Wen X, Gu G, Li Q, et al. Comparison of open-source cloud management platforms: OpenStack and OpenNebula[C]//Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on. IEEE, 2012: 2457-2461.

[7] Eucalyptus www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html.

[8] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 164-177.

[9] Kivity A, Kamay Y, Laor D, et al. KVM: the Linux virtual machine monitor[C]//Proceedings of the Linux symposium. 2007, 1: 225-230.

[10] Borthakur D. The Hadoop distributed file system: Architecture and design[J]. Hadoop Project Website, 2007, 11(2007): 21.

[11] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2.

[12] Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS operating systems review. ACM, 2003, 37(5): 29-43.

[13] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[14] Burrows M. The Chubby lock service for loosely-coupled distributed

systems[C]//Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, 2006: 335-350.

[15] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4.

[16] Wilder B. Cloud architecture patterns: using microsoft azure[M]. " O'Reilly Media, Inc.", 2012.

[17] Leavitt N. Will NoSQL databases live up to their promise?[J]. Computer, 2010, 43(2): 12-14.

[18] 王胜文. 基于软件的故障注入方法研究[D]. 哈尔滨: 哈尔滨工业大学, 2005.

[19] Arlat J, Crouzet Y, Karlsson J, et al. Comparison of physical and software-implemented fault injection techniques[J]. Computers, IEEE Transactions on, 2003, 52(9): 1115-1133.

[20] Segall Z, and Lin T, “FIAT: Fault Injection Based Automated Testing Environment,” Proc. 18th Int’l Symp. Fault-Tolerant Computing,pp. 102-107, June 1988.

[21] Han S, Rosenberg H, and Shin K, “DOCTOR: An Integrated Software Fault Injection Environment,” Technical Report, Univ. of Michigan, 1993.

[22] Kanawati G, Kanawati N, and Abraham J, “FERRARI: A Tool for the Validation of System Dependability Properties,” FTCS-22,Digest of Papers, pp. 336-344, 1992

[23] Kao W, Iyer R, and Tang D, “FINE: A Fault Injection and Monitoring Environment for Tracing the UNIXSystem Behavior Under Faults,” IEEE Trans. Software Eng., vol. 19. no. 11, Nov. 1993.

[24] Le M, Gallagher A, and Tamir Y, “Challenges and Opportunities with Fault Injection in Virtualized Systems,” in 1st Int. Workshop on Virtualization Performance: Analysis, Characterization, and Tools,2008.

[25] Ju X, Soares L, Shin K, Ryu K, and Silva D, “On fault resilience of OpenStack,” in Proceedings of the 4th annual Symposium on Cloud Computing, 2013, pp. 1–16.K.

[26] Benz and Bohnert T, “Dependability Modeling Framework: A Test Procedure for High Availability in Cloud Operating Systems,” in Vehicular Technology

Conference (VTC Fall), 2013 IEEE 78th, Sept.2013, pp. 1–8.

[27] Souza D,Matos R, Araujo J, Alves V, and Maciel P, “A Tool for Automatic Dependability Test in Eucalyptus Cloud Computing Infrastructures,” Computer and Information Science, vol. 6, no. 3, May.2013.

[28] ChaosMonkey. http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html.

[29] Varia J. Architecting for the cloud: Best practices[J]. Amazon Web Services, 2010.

[30] De Simone L. Towards Fault Propagation Analysis in Cloud Computing Ecosystems[C]//Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on. IEEE, 2014: 156-161.

[31] Gunawi H S, Do T, Joshi P, et al. FATE and DESTINI: A framework for cloud recovery testing[C]//Proceedings of NSDI’11: 8th USENIX Symposium on Networked Systems Design and Implementation. 2011: 239.

[32] 冯刚. 面向云计算平台的虚拟机故障注入工具研究与设计[D]. 哈尔滨工业大学, 2013.

[33] 麻彦东. 面向虚拟化系统的故障注入平台的研究与设计[D]. 哈尔滨工业大学,2015.

[34] 赵志龙. Hadoop 容错能力测试平台的设计与实现[D]. 哈尔滨工业大学, 2013.

[35] Apache Storm. https://storm.apache.org.

[36] 张贵麟. 面向高可用服务的虚拟机动态迁移机制研究[D]. 哈尔滨工程大学,2014

[37] Krishnakumar R. Kernel korner: kprobes-a kernel debugger[J]. Linux Journal, 2005, 2005(133): 11.

[38] 使用 Kprobes 调试内核. http://www.ibm.com/developerworks/cn/linux/l-kprobes.html

[39] Jones M T. Access the Linux kernel using the /proc filesystem[J]. IBM developerWorks, 2006.

[40] YANG G, CHEN S. Research on Linux firewall based on Netfilter/Iptables [J]. Computer Engineering and Design, 2007, 17: 022.

[41] 丁滟, 富弘毅, 李宇卓. Linux 下 VFS 层 Rootkit 技术研究[J].计算机工程, 2010, 36(8):161-164.

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《云计算系统故障注入平台的研究与设计》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：  日期： 2016 年 6 月 30 日

学位论文使用权限

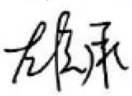
学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

（1）学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；（2）学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；（3）研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：  日期： 2016 年 6 月 30 日

导师签名：  日期： 2016 年 6 月 30 日

致　谢

在这两年半的时间里，我首先感谢的是张展老师，无论是实验的日常还是项目，都离不开张老师忙碌的身影。张老师对我们都很信任，在项目上给我们很大的发挥空间，从实验室的项目中我学到了很多的技术和能力，无论是技术的细节，还是和大家的合作沟通。

然后我要感谢我的导师左德承教授，左老师对我们严格要求，在科研上一丝不苟，您的忘我的工作态度让我受益终身。在论文的开展工作遇到困难时，左老师总能给出建设性的意见。感谢我的导师两年来对我的关心。

在实验室，最难忘记得身边的兄弟姐妹和师兄师姐，你们让我在实验室的生活变得丰富多彩，充实而又意义。在项目上我要感谢王旭师兄，李文浩师兄，还有已经毕业的麻彦东师兄，胡军杰师兄，你们让我能快速的进入项目工作中。我还要感谢其余的各位师兄师姐和我同届的同学。我们一起漫步在通往南苑的小路上。我还要感觉我的寝室和我周围寝室的同学。

最后，我要感谢我的家人，你们来自远方的挂念让我感到家的温暖。

最后，感谢各位专家与教授的审查，你们辛苦了。