单位代码：10293 密　级：公开

南京邮电大学

专 业 学 位 硕 士 论 文

论文题目：基于MySQL的云数据库设计与实现

学　　号　1213012235

姓　　名　牛小宝

导　　师　钱学荣

专业学位类别　工程硕士

类　　型　全　日　制

专业（领域）　电子与通信工程

论文提交日期　2016年4月

The design and implementation of cloud database base on MySQL

Thesis Submitted to Nanjing University of Posts and Telecommunications for the Degree of

Master of Engineering

By

Xiaobao Niu

Supervisor: Prof.Xuerong Qian

April 2016

南京邮电大学学位论文原创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京邮电大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

本人学位论文及涉及相关资料若有不实，愿意承担一切相关的法律责任。

研究生签名：_____ 日期：_____

南京邮电大学学位论文使用授权声明

本人授权南京邮电大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档；允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索；可以采用影印、缩印或扫描等复制手段保存、汇编本学位论文。本文电子文档的内容和纸质论文的内容相一致。论文的公布（包括刊登）授权南京邮电大学研究生院办理。

涉密学位论文在解密后适用本授权书。

研究生签名：_____ 导师签名：_____ 日期：_____

摘要

云数据库是传统数据库与云计算技术相结合的产物，它能够提供数据库资源的虚拟化，并且具有按需使用、动态扩容、高可用性等优点。近几年云数据库技术飞速发展，越来越多的企事业单位、组织将数据库转移至云端，因此对构建云数据库的需求也在不断增加。目前市场上的云数据库大多数是在公有云的环境上构建，但私有云数据库同样有很大的市场需求。为此，本文通过研究OpenStack和集群管理技术，分析SaltStack自动化部署和双机热备高可用架构在云数据库设计中的可行性，提出了一种在私有云环境上构建云数据库的解决方案，方案具有较好的易用性和可靠性，并且支持关系型数据库MySQL引擎。

本文首先介绍了云计算和云数据库的相关概念以及发展现状，并对构建云数据库过程中使用到的关键技术进行了阐述和分析，通过分析云数据库的实际使用场景，制定了云数据库项目的功能性和可靠性方面的需求，并给出详细的需求说明。

然后对云数据库系统进行模块划分，从物理架构和逻辑架构两个方面去制定每个模块的部署方案和逻辑设计，为了提高系统的易用性和可靠性，提出了基于SaltStack的自动化部署设计和基于共享存储的高可用集群架构设计。

接着给出本次云数据库项目中主要功能需求的实现，详细解释了功能类的设计和数据库表结构设计，设计并给出了每个功能的工作流程以及具体实现方式。最后通过搭建基于MySQL的云数据库测试环境，对项目中的主要功能和高可用方案进行了测试，验证了云数据库系统的功能性、易用性和可靠性。

关键词:云计算， 私有云， 云数据库， MySQL， SaltStack

Abstract

Cloud database is a product of traditional database technology combined with cloud computing, it can provide database resource virtualization, and has on-demand, dynamic expansion, high availability. In recent years, the rapid development of cloud database technology, more and more enterprises, institutions, organizations move databases to the cloud, so the demand for building cloud database is also increasing.Cloud database currently on the market most of the public cloud environment to build, but a private cloud database also has a great market demand. In this paper, by studying OpenStack and cluster management technology, analysis SaltStack automated deployment and hot standby high availability architecture feasibility cloud database design, proposed to build a cloud database solution in a private cloud environment, the program has better ease of use and reliability, and supports MySQL relational database engine.

Firstly, in this issue, it introduces the concept of cloud computing and cloud database and the development of status quo, analysis of key technologies to build cloud database. By analyzing the actual usage scenarios cloud database, developed a demand cloud database functionality and reliability, and gives a detailed description of the requirements.

Secondly, divide the cloud database system module, from the physical structure and logical structure two aspects to develop a deployment plan for each module and logic design. In order to improve the usability and reliability of the system, the author proposes to SaltStack-based automated deployment design and based on the shared storage high availability cluster architecture.

Thirdly, given the realization of the cloud database project the main functional requirements, design and gives each function workflow and specific implementation. Lastly, by building the cloud database test environment based on MySQL, test the project's main features and high-availability solutions, verify the functionality and ease of use and reliability of the cloud database system.

Key words: Cloud Compute, Private Cloud, Cloud Database, MySQL, SaltStack

目录

第一章 绪论

1.1 课题背景

自云计算的解决方案诞生以后，企事业单位通过对云计算的应用，免除了传统业务中对软件硬件资源的购买，部署以及维护的费用和烦恼，很大程度上降低了企事业单位管理以及运营的成本[1]。人们越来越重视将商业应用和企事业内部管理转移至云计算的模式中，所以希望将更多的资源以云计算的方式进行虚拟化拓展，如主机、存储、网络、数据库、负载均衡等基础服务设施。其中数据库是应用中不可缺少的基础资源，但是传统数据库的管理运维需要耗费大量的资金人力，并且安全性也比较薄弱，而通过使用云数据库，企事业只需要通网络连接至云端资源，就可以按需使用数据库资源，无需采购大量设备，并且数据的安全性更有保障。因此无论是在公有云还是私有云中，对云数据库的研究都能够推动企事业单位高效率，低成本的工作运营，具有重要的意义。

本论文中对云数据库研究主要是基于某通信技术有限公司的私有云平台进行的设计，此云平台已投入商业用途，并应用于多家政府机构和事业单位。云数据库后端采用的是应用最为广泛关系型数据库RDS，即MySQL。笔者参与了云数据库的研发流程，本文将对云数据库研发过程中使用到的技术背景、产品的需求分析以及关键技术的实现进行介绍。

1.1.1 云计算

云计算（Cloud Computing），是一种基于互联网计算的方式，利用这种方式，可以按照不同的需求分配共享的软硬件资源和信息给其他的计算机和设备使用[2]。在2006年8月9日的搜索引擎大会上（SES San Jose 2006），Google的首席执行官Eric Schmidt第一次给出了云计算的概念。

云计算是通过数量众多的分布式计算机来提供强大的计算能力[3]，因此比本地计算机或者服务器的计算能力要提升很多。如果把计算能力比作一种商品可以进行流通，如电、水等，那么普通计算向云计算的转变就像原始的单台发电机模式向发电厂集中供电模式的转换，用户可以通过互联网对这些计算资源按需索取，按量计费。

云计算主要有以下几个特点：

（1）超大规模：很多云计算中心都具有相当大的规模，比如：谷歌的云计算中心拥有上百万台的服务器资源，而IBM、亚马逊[4]等公司的云计算规模也丝毫不逊色。云计算中心通过

对这些服务器集群进行资源的整合和管理，能够为用户提供强大的计算能力和存储能力[5]。

（2）抽象化：用户可以在任何位置使用终端通过网络获取云计算服务，用户请求的资源并不是实体，而是在云端通过虚拟化技术为用户提供服务，应用使用云端的资源运行，用户无需关注应用运行的具体位置，大大简化了应用使用的复杂度。

（3）高可靠性：云计算往往采用多种高可用方案来保证服务的高可靠性，如双机热备，服务器心跳检测，多副本容错等多种方案。

（4）高可扩展性：云资源通常可以根据用户的实际使用情况进行动态的调整和拓展（例如云硬盘的扩容），并且依赖于云计算中心本身的大量的集群资源，可以最大限度的满足应用和用户对资源需求的大规模增长。

（5）通用性：云计算中心对业界大多数的主流应用都可以兼容运行，并且可以同时运行多个不同类型的应用并能够保证应用的运行质量。

（6）自动化：对于云计算中的多种资源应用，大多都能够通过一定的方式实现远程自动化部署和管理，自动化的方式从很大程度上降低的运维的成本和复杂度。

（7）低成本：云计算中心的大规模集群资源可以很大程度上的提升计算资源的利用率和工作效率，降低应用和服务的成本。

（8）按需服务：云计算是将资源的池化的一种虚拟技术，用户可以按照自己的需要购买资源，这样就避免用户为无用的资源买单[6]。

（9）完善的运营机制：云服务提供商提供专业的运维团队帮助管理用户信息，按照严格的权限管理策略保护用户数据的安全，提供先进、庞大的数据中心保存用户的数据。

云计算主要提供三个层次的服务，分别为基础设施即服务（IaaS，Infrastructure as a Service）、平台即服务（PaaS，Platform as a Service）和软件即服务（SaaS，Software as a Service）[7]。

（1）基础设施即服务（IaaS）：用户可以通过互联网可以从云端的基础设施资源（物理设备）获得服务。例如：硬件服务器租用。

（2）平台即服务（PaaS）：是提供解决方案和运算平台的服务，以软件即服务的模式提交给用户，是介于IaaS和SaaS之间的服务层。例如：软件个性化定制开发。

（3）软件即服务（SaaS）：是一种软件交付的模式，在云端对软件以及软件的数据库进行统一的托管，用户使用一个精简的客户端就可以通过WEB浏览器来访问托管的软件。

云计算在商业运用中有三种常见的模式，分别为：公有云、私有云、混合云，不同的模式运用与不同的场景，下面简单介绍这三种云计算模式：

（1）公有云：是目前运用相对主流的云计算模式，公有云通过极其庞大的集群资源对公众开发它的云计算服务，因其强大的请求处理能力和较低的使用成本，深受广大用户的欢迎。

（2）私有云：主要是企事业单位内部使用的云服务，这种云模式集群资源相比公有云较小，但是由于不对公众开发，且是针对企事业单位定制，所以更加吻合企业内部的使用方式，数据的安全性以及服务的质量都能得到更好的保障。

（3）混合云：由两个或者更多云计算系统模式组成云计算基础资源设施，这些云计算系统包含了私有云、公用云等。并在私有云和公有云各自的优点之间进行一定的权衡和融合。

1.1.2 云平台

随着云计算的逐渐发展以及技术的日益成熟，各种云平台应运而生。云平台，即云端的资源和服务的管理平台，它基于三个层次云计算服务，将资源、服务整合后以Web服务的形式提供给用户，供用户使用。

目前国外较为著名的云平台有OpenStack、Eucalyptus、OpenNebula、AbiCloud、Amazon EC2[8]等，其中尤以OpenStack在全球关注度最高且技术发展最快。OpenStack是由美国国家航空航天局（NASA）与美国云服务提供商Rackspace公司于2010年9月共同发起并推动的一个非盈利性的开源云计算项目。它是基于完全开源的思想，目的是推动实现云计算开放性、无边界性的技术实质，为云计算管理平台领域带来更加开放、自由与灵活的构建方法，致力于提供规模化、灵活扩展易部署且功能丰富的全开源模式平台，协助企事业单位实现公共云和私有云服务[9]。

OpenStack项目自发起后发展迅速，每年发布两个版本，节奏清晰，路标明确。在最新版本主要由多个组件组成：Nova、Swift、Glance、Keystone、Horizon、Cinder、Cellometer、Heat

其中Nova为核心组件，实现弹性计算服务；Swift组件提供对象存储服务；Glance组件用于管理镜像模板；Keystone组件提供认证服务和权限管理；Horizon组件提供Web管理界面；Cinder组件提供块存储服务；Cellometer组件为计量测量组件，用于性能监控和度量；Heat组件为业务模块，用于自动创建或者拆卸基础设施。

OpenStack基于REST（Representational State Transfer，表述性状态转移）架构风格实现，能够易用和简洁的将服务提供给调用者。

目前国内较为著名的云平台有阿里云（aliyun.com）、百度云（Baidu Cloud）、华为云（HUAWEI Cloud）。

（1）阿里云：阿里巴巴集团旗下的云计算品牌，创立于2009年，是目前国内最大的公有

云计算服务提供商，现在用户数量庞大的用户，用户机会遍布各个领域。

（2）百度云：百度推出的云存储服务平台，提供多元化的数据存储服务，用户可自由管理网盘存储文件，目前已覆盖主流PC和移动设备的操作系统。

（3）华为云：华为公司推出的公有云管理平台，创立于2011年，华为云主要面向企事业单位提供服务和解决方案，云产品主要有弹性云服务器，云硬盘，对象存储等，解决方案有金融云，媒资云，政务云等。

1.1.3 云数据库

云数据库就是将数据库与云计算结合在一起，将数据库部署到一个虚拟计算环境中进行使用。用户使用云数据库时，可以实现按需付费，弹性扩容，动态升级等功能。云数据库往往采用高可用的设计方案，保证用户使用数据的可靠性[10]。云数据库的优点如下：

（1）用户可以根据实际使用情况定制自己需要的数据库，按照存储容量，最大连接数等需求付费，无需再为无用的资源买单。

（2）云数据库的高可用架构，可以实现故障切换，保证了数据库的可靠性，大大降低了由于不可预知的故障宕机带来的巨大风险。

（3）云数据库通常可以远程进行参数修改（性能调优），弹性扩容等功能，大大的降低的运维的成本。

（4）云数据库为用户提供了灵活的备份策略，以及实例的监控告警，最大限度的保证数据库实例的健康运行。

1.2 本文选题来源和主要工作

目前国内云平台的发展逐渐成熟，其中比较著名的百度云、阿里云和华为云等提供的均是公有云服务，而私有云平台在国内的云平台模式中所占的比重较低，但是不可否认的是私有云平台具有很大的发展潜力，近年来，各企事业单位、组织对于构建自己的私有云数据库的需求也在不断的增加。

本课题的选题来源于作者所在项目组开发的私有云管理平台系统，系统基于云计算的架构、管理、运营，实现了IaaS层的基础设施的虚拟化和管理。云数据库是私有云管理平台系统中的应用之一，云数据库支持的各种功能以Web Service的形式呈现，实现了数据库的实例创建，删除，启动停止，备份恢复，参数修改，实例监控等功能，为第三方开发者提供Paas层的服务，具有高可用性，多租户，高拓展性等特点。

本文介绍了系统开发所依赖的关键技术，以及这些技术在系统实际开发过程中的应用 。制定了私有云管理平台在IaaS层需提供的云数据库的需求，包括数据库服务器池，实例的创建删除，实例的备份恢复，监控告警等，介绍了系统架构与逻辑设计、系统模块划分和系统部署设计，最后详细介绍了云数据库的设计与实现。

1.3 本文的组织结构

本文共分为七章，具体安排内容如下。

第一章 绪论

介绍本文的相关背景，包括云计算和云数据库的概念、特点、服务层次、模式以及国内外云平台软件的发展现状。介绍本文的选题来源和主要工作。

第二章 技术背景

介绍系统在开发过程中依赖的关键技术和思想。

第三章 系统需求分析

介绍系统目标，分析系统的功能需求，介绍系统的主要业务流程。

第四章 系统体系架构

在系统需求分析的基础之上，介绍系统的框架结构设计、模块结构设计和部署设计。

第五章 系统功能详细设计与实现

详细介绍由笔者负责的云数据库系统的主要功能的设计与实现，包括功能设计与数据库表设计、模块的关键实现等。

第六章 系统的测试与验证

对系统进行功能性测试和验证，测试了系统的主要功能：添加物理服务器池、创建云数据库实例和系统的高可用性。

第七章 总结和展望

总结云数据库现阶段的优缺点，提出下一步的工作展望。

第二章 技术背景

2.1 OpenStack

OpenStack是一个开源的云计算管理平台项目，它是由多个不同子组件间的相互协作完成向用户提供计算，存储和网络等服务资源的任务。Openstack包含的子组件主要有：计算组件Nova、镜像管理组件Glance、虚拟网络管理组件Neutron、块存储组件Cinder、对象存储组件Swift、身份认证组件Keystone和控制面板组件Horizon以及消息中间件RabbitMQ等[11]，OpenStack设计时尽量降低每隔组件之间的耦合度，所以每个组件也可以单独的向用户提供资源服务，各个组件之间的关系如图2.1所示。

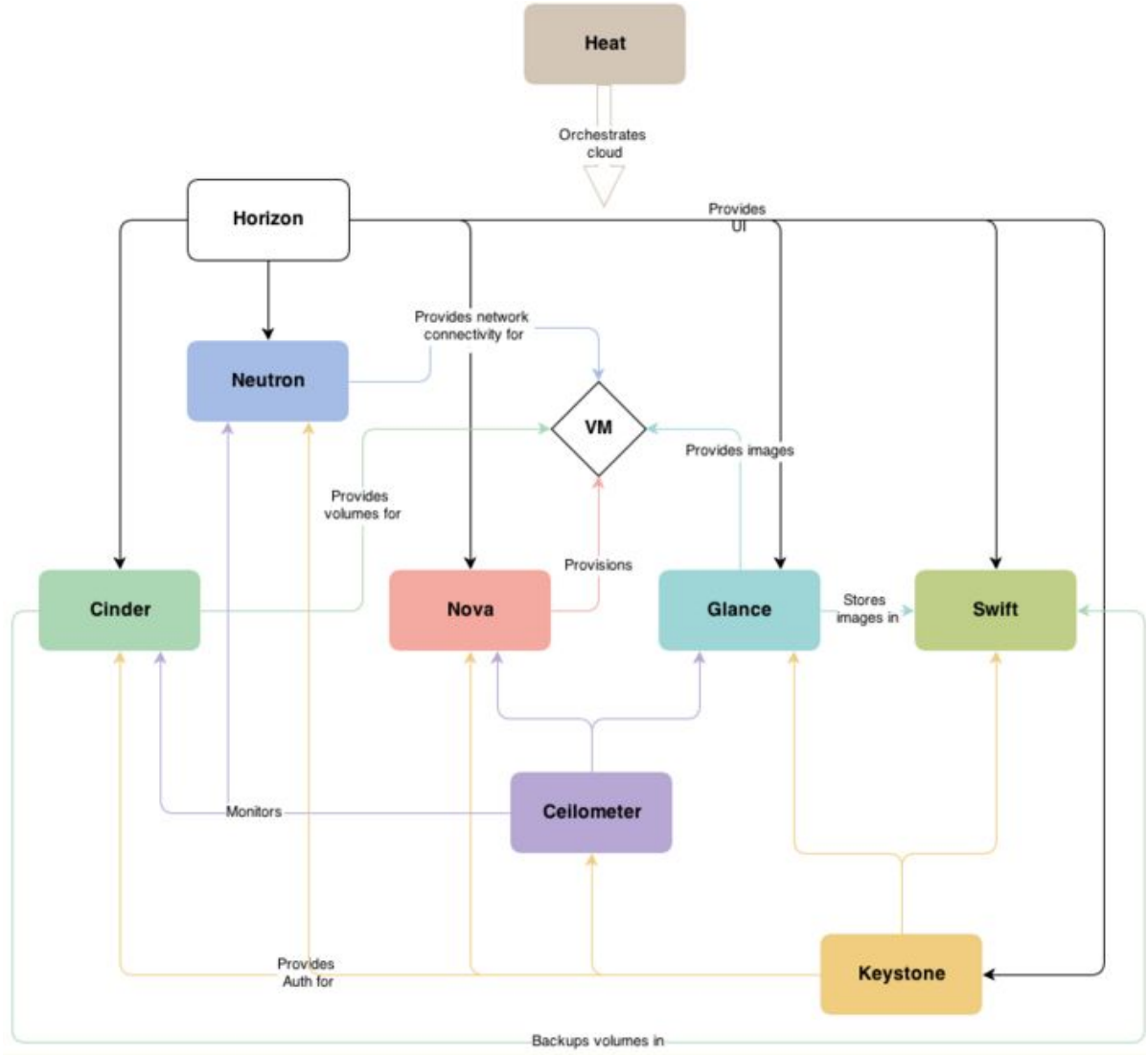

图2.1 OpenStack组件关系图

2.1.1 Nova

Nova是OpenStack中最重要的一个组件，其实现起来也要比其他组件复杂，Nova中包含多个不同的服务，这些服务用于管理虚拟机实例，贯穿虚拟机实例的整个生命周期，包括虚拟机实例启动、停止、重启、终止等操作[12]。

（1）Nova-api：是Nova的核心模块，Nova模块是通过Nova-api对外向用户提供服务。Nova-api提供了WSGI（Web Server Gateway Interface）服务，它可以将从WEB发送过来的请求路由到相应服务（service）中的具体方法，这些具体方法会完成相关服务的请求。它是通过消息队列与别的组件进行交互。

（2）Nova-compute：是相当重要的守护进程，管理着虚拟机实例的创建和终止等操作[13]。其基本原理就是从消息队列中接收消息请求，并且执行相应的系统操作完成请求，并将数据库的状态进行更新。在一个典型的云部署产品中会有多个Nova-compute，实例部署在哪个Nova-compute上取决于使用的Nova-schedule算法。

（3）Nova-sheduler：根据当前计算节点的剩余资源进行一定的权值分析，找出最优的计算节点来承担计算能力。从可获得的资源池中利用哪些计算、网络、存储服务器等资源依赖于具体的算法，算法需要考虑的因素很多，包括：负载、内存、有效范围内的物理距离、CPU等。目前调度算法的实现支持插件方式扩展，用户可以根据实际情况定义自己的调度器。

（4）Nova-volume：负责为虚拟机实例分配额外持久化的存储，可以管理计算实例创建，连接和分离持久卷。volume可以使用来自不同卷提供商供应的卷，比如AoE、iSCSI。持久化使得通过连接该卷到同一个实例或者连接到另一个实例，卷中的数据仍然可用。实例在生命周期中的堆积起来的任何贵重数据都将被写到卷里面去，这是为了在以后访问这些数据。

（5）Nova-network：负责为虚拟机分配网络，并对这些网络进行管理，使外部的终端设备可以通过网络访问虚拟机。它从消息队列中接收有关网络的相关请求，然后执行请求完成相关的网络操作。网络任务包括：分配IP地址、为项目配置VLAN、为计算节点部署安全组和配置网络等。一般来说也是有一个公网IP和私网IP。私网IP用来和其他实例进行通信和交互，公网用来和外部通信和交互。

（6）消息队列服务器：消息队列服务器主要用于不同组件之间的相互通信，Nova中的组件扮演消息生产者和消费者的角色。通过消息队列的消息发送有两种方式：同步调用和异步调用[14]。OpenStack中采用的消息队列为RabbitMQ。

（7）SQL database：大多数的Nova服务使用一个SQL database来存储信息，database通常在控制节点上运行。OpenStack的可拓展性保证了对多种数据库的支持。目前使用最广泛的是

数据库有MySQL、PostgreSQL和MariaDB等。

上面介绍的这几个组件都是Nova内部的组件，为Nova提供内部的交互服务。实际工作中，Nova还需要和其他OpenStack的组件进行交互，共同为虚拟机实例的运行提供所需的资源，比如Nova需要调用Glance-api完成虚拟机实例镜像文件的存储与读取，需要与Cinder组件交互为虚拟机实例挂载持久卷等，下面将介绍OpenStack其他的重要组件。

2.1.2 存储方式（Swift，Glance，Cinder）

Glance的镜像服务包括查询，注册和恢复虚拟机镜像。Glance具有RESTful风格的API，可以支持查询虚拟机镜像的元数据，以及实际镜像的恢复。Glance模块可以有三个主要部分组成：

（1）RESTAPI：主要负责与用户的交互，API 向上以RESTful风格向外提供Glance的功能，向下可以调用Glance Domain控制器指定每一层完成相应的请求。

（2）Registry：这一层主要与DAL（Database Abstraction Layer）进行交互，接收来自于API的请求，供API进行调用以维护虚拟机镜像文件的状态。

（3）Glance Store：用于组织管理Glance与各种数据存储之间的相互作用，Glance是通过插件的方式支持众多的后端存储方式，包括：Filesystem，Swift，S3，Ceph，Sheepdog等。

Cinder是OpenStack Block Storage项目名称，为OpenStack提供块存储的服务[15]。它可以通过Nova服务进行调用并为最终用户提供存储资源。Cinder就是一个虚拟化的块存储资源池，并通过自由的API对外提供服务，使用者无需关注实际的存储类型或设备种类。

Cinder的内部架构有三个重要组成部分：

（1）Cinder-api：主要负责对外提供REST风格的API，用户或者其他组件通过API对Cinder服务进行调用。

（2）Cinder-scheduler：主要负责任务调度，并分配存储资源。Scheduler有一套任务调度策略的算法，可以选择出最优的Cinder-volume解决来处理用户的任务请求。

（3）Cinder-volume：该服务主要用来管理存储空间，处理来自于Cinder数据库的读写等操作请求。volume服务运行在存储节点上，因此每一个存储节点有运行一个volume service，通过联合多个存储节点构成一个存储资源池。Cinder-volume通过插件形式提供对多种后端存储形式的支持，如LVM、Sheepdog、NFS、RBD 等。

Swift是OpenStack Object Store项目的名称，它致力于提供一个高可用性，分布式的，最终一致的对象存储。用户们可以使用Swift对大规模的数据进行高效，安全，低廉的存储。对

非结构化无限增长的数据，Swift是理想的存储方式。Swift组件主要由以下几个重要部分组成：

（1）Rings：该服务可以用来确定数据应该存储在哪个集群中，有不同类型的Ring用于区分Account database，container database和个人的对象存储，但是每种Ring的工作方式是相同的。

（2）Proxy Server：该服务对外提供Swift服务的API，API采用的无状态的REST风格的请求协议。Proxy Server会根据Ring的信息进行服务地址的查找到，并将请求发送至相应的Account Server，Container Server和对象服务进行处理。

（3）Storage Server：该组件主要为Swift提供真正的存储服务，在Swift 中有三类存储服务：账户服务：管理由Object Server中定义的账户信息；容器服务：管理容器与它对应的对象之间的映射关系；对象服务：管理存储节点中的对象信息[16]。

2.1.3 Keystone

Keystone（OpenStack Identity Service）是Openstack项目中提供身份验证，令牌服务，注册登记等服务，服务的对象是OpenStack项目中的各个组件[17]。Keystone实现了OpenStack项目中有关身份验证的API，为了实现这一功能，Keystone团队还提供了一个类似于Python客户端库的WSGI中间件。在OpenStack中，任何服务之间的调用都需要经过Keystone的身份验证，验证通过才能够根据在Keystone注册的服务访问URL来获取目标服务。

2.1.4 Neutron

Neutron最初是由Quantum演变而来，它的设计是为了给OpenStack各接口设备（如vNICs）提供“作为一种服务的网络”，Neutron是由OpenStack其他服务（如Nova）共同管理调用[18]。为了更明确的描述实际情况的网络资源，Neutron抽象出了三种模型：网络、子网、端口。

OpenStack通过Neutron服务把传统的本地网络管理的部分功能通过远程的形式提供给了租户，租户利用它可以虚拟出自己的网络，并用其进行划分子网、创建路由等操作。通过Neutron提供虚拟网络功能，就可以将基础网络服务进行拓展，向外部提供额外的网络服务，比如用户使用网络服务去构建自己工作使用的虚拟网络中心。虽然Neutron提供的完善易用的API，以及多种多租户环境下的虚拟网络模型，但使用Neutron创建虚拟网络时仍需要像部署物理网络一样进行基本的规划和设计。

2.2 Saltstack

SaltStack是由Python语言开发，具有自动化配置管理和远程命令并行执行等功能的工具，简单易部署，可伸缩的足以管理成千上万的服务器和足够快的速度控制。它能够简化系统管理员对基础设置的管理操作，包括安装软件、配置参数和服务的启动停止等，从而实现复杂的功能和系统管理[19]。

SaltStack支持Windows，Linux等多种主流操作系统。SaltStack的架构分为Salt-Master端和Salt-Minion端，两者之间采用长连接的方式，它支持的配置管理的工作模式有两种：Salt-Master端主动进行推送和Salt-Minion端定时获取配置。在远程命令并行执行方面，SaltStack的命令回显美观易读，还具备很多主流的命令编排工具，如：MCollective，FUNC等，而且自身对大多数的日常执行模块也都支持。它使用State模块获取对应集群的状态和相应状态之间所依赖的关系，比如文件，定时任务，用户，软件安装包和系统服务等。

SaltStack具有简单易拓展和高度模块化的设计，它是根据管理员工作中的不同需求来进行模块的划分，通过模块间的配合完成特定的任务，主要模块有执行模块和状态模块。

2.2.1 SaltStack 执行模块

SaltStack系统的基础核心功能是由执行模块构成的，在Salt-Minion节点上，执行模块可以用来执行多种命令，包括查看Salt-Minion节点系统相关的信息、软件包的安装、脚本命令的执行等。SaltStack是通过高性能的消息中间件ZeroMQ向Salt-Minion传递消息，消息以广播的形式传递到相应的目标Salt-Minion节点，目标Salt-Minion执行消息命令后，同样通过中间件ZeroMQ将执行结果返回至Salt-Master。

系统管理员通过执行salt '<target>' <function> [arguments]命令来调用相应的模块对指定的Salt-Minion节点执行相应的功能。target即目标节点的匹配方法，arguments为函数中的可选参数列表，function是执行模块对应的内部函数方法名。

2.2.2 SaltStack 状态模块

SaltStack的状态模块（Salt State）负责SaltStack系统的状态配置管理，状态管理程序可以让系统保持或者达到预先定义的状态，它能够依据用户定义状态的描述，自动进行软件的安装、重启或者打开服务、将配置文件传递的指定的存储位置并可以对其进行监控[20]。

使用State模块进行状态管理，可以极大的降低管理集群的难度，管理员可以轻松的对数

百台甚至数万台集群服务器的进行配置管理。实际应用中，可以将状态管理的配置文件通过版本控制工具进行管理，如git，svn等，这样能够更好的管理配置文件的变更。

2.3 消息中间件

2.3.1 RabbitMQ

（1）RabbitMQ介绍

RabbitMQ是一个开源的AMQP（Advanced Message Queuing Protocol）实现，它是一个面向消息设计的中间件，服务器端采用Erlang语言编写，主要用于各组件见的消息通信，消息的发送者或者使用者只需要关注消息本身而不需要知道对方的存在[21]。RabbitMQ支持多种客户端，如：Python、Ruby、Java等，支持AJAX。 在分布式系统中，RabbitMQ在存储转发消息，在易用性、可扩展性、高可用性等方面具有出色的表现。

对RabbitMQ和消息中一些常用的专有名词进行如下介绍：

1) 生产（Producing ）：在RabbitMQ中即为发送，发送消息的程序就是一个生产者（producer）。

2) 队列（Queue）：用来进行消息的中转，消息通过应用程序和RabbitMQ进行传输，它们能够只存储在一个队列中。队列存储的消息数目没有任何限制，基本上是一个无限的缓冲。多个生产者能够把消息发送给同一个队列，同样，多个消费者也能够从同一个队列中获取数据。队列的属性有持久性、自动删除、惰性、排他性。

3) 消费（Consuming）：即为从队列从获取消息进行处理。一个消费者（consumer）就是一个等待获取消息的程序。

4) 交换器（Exchange）：交换器在RabbitMQ中起到消息路由的作用，生产者将消息发送到交换器，由交换器将消息路由到一个或多个队列中（或者丢弃）。

几个模块之间的关系如图：

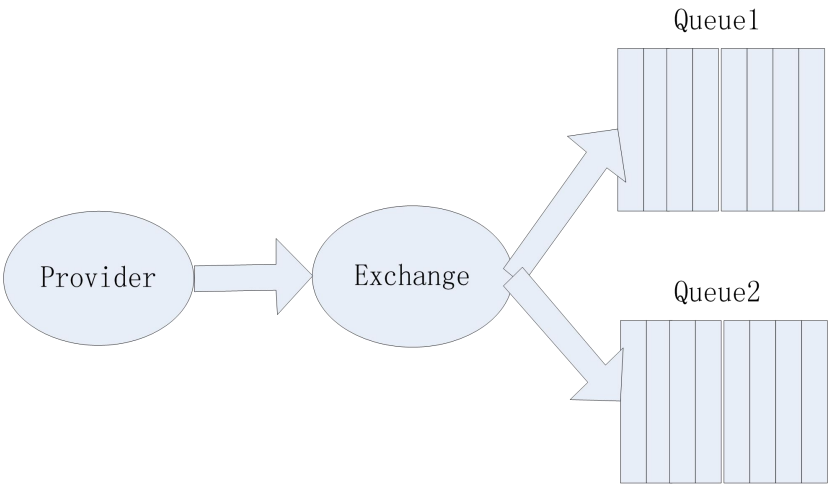

图2.2 RabbitMQ关系图

（2）RabbitMQ在OpenStack中的应用

在Openstack中，组件之间对RabbitMQ使用基本都是“Remote Procedure Calls”的方式。每个Nova服务（比如计算服务、存储服务等）初始化时会创建两个队列，分别名为“NODE-TYPE.NODE-ID”，“NODE-TYPE”，其中NODE-TYPE是指服务的类型，NODE-ID是指节点名称[22]。

RabbitMQ作为OpenStack的中间件节点使用时，其交互过程可以抽象成下图：

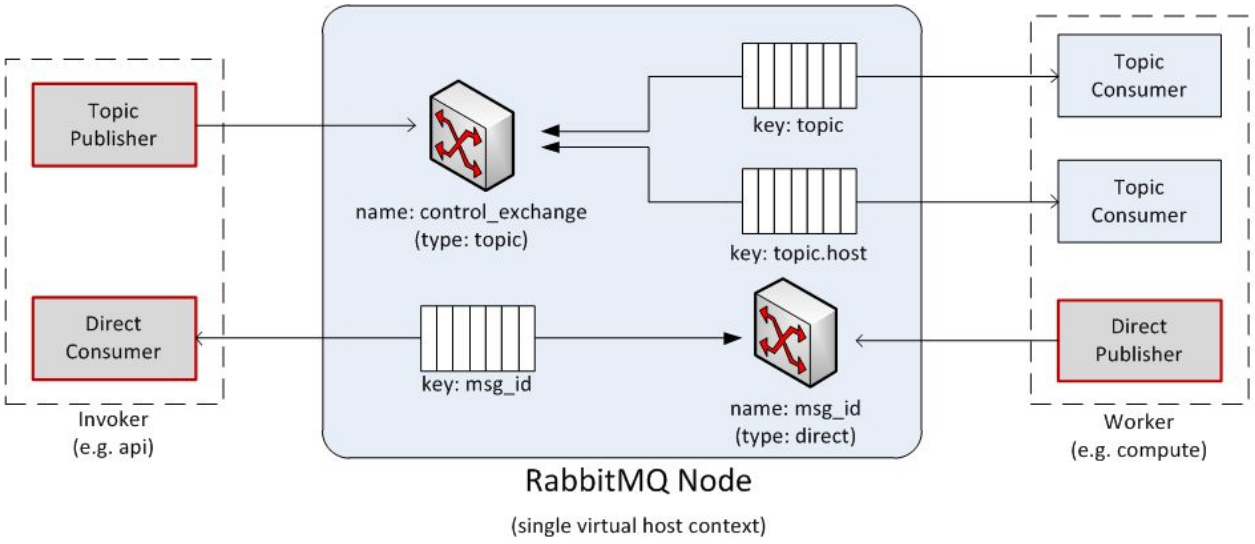

图2.3 RabbitMQ工作流程

每个生产者服务都会有一个相应的topic类型的exchange与之对接，每个exchange上又会绑定两个队列，消费者服务从不同的队列中接收不同类型的消息。如果消息的生产者想要关注消息消费后的结果，则需要监听另一个队列，该队列绑定在一个类型为direct的exchange上[23]，消费者完成消息处理后，会把返回结果发送至这个direct类型的exchange上。

2.3.2 ZeroMQ

（1）ZeroMQ介绍

ZeroMQ是一个简单好用的高性能异步消息库，它虽然像一个socket library，但却担任一个框架的角色。ZeroMQ提供了类似Socket的API接口，使Socket编程变得更加的具有高性能和简洁性。它与Socket的区别在于：传统的Socket只支持端到端的1:1通信方式，而ZeroMQ却可以是N:M的关系。ZMQ提供了三个基本的通信模型，主要包括请求应答模型（ROUTER/REQ）、发布订阅模型（PUB/SUB）和管道模型[24]。ZeroMQ还具有如下多个特点：

1) 可以用任何编程语言和平台连接ZeroMQ；
2) 提供进程内、TCP、IPC和多播方式的消息通讯；
3) 具有多种通信模式，如pub-sub，push-pull和router-dealer；
4) 以一个很小的库，提供高速异步的I / O引擎；
5) 支持多种模式的语言和平台；
6) 背后拥有一支庞大且活跃的交流平台资源；

（2）ZeroMQ在SaltStack中应用

SaltStack的底层网络架构采用的是ZeroMQ，Salt利用ZeroMQ灵活高效的patterns，使Salt网络拓扑变得非常灵活高效。利用PUB/SUB实现了高效的远程执行指令下发机制，利用ROUTER/REQ实现认证及异步的远程执行结果返回，利用DEALER/REP实现多进程任务处理机制，利用PULL/PUB实现Event BUS，使其他或第三方应用可以快速的使用PUB/SUB接收到Event BUS上的消息，SaltStack中Master和Minion与ZeroMQ的关系如图：

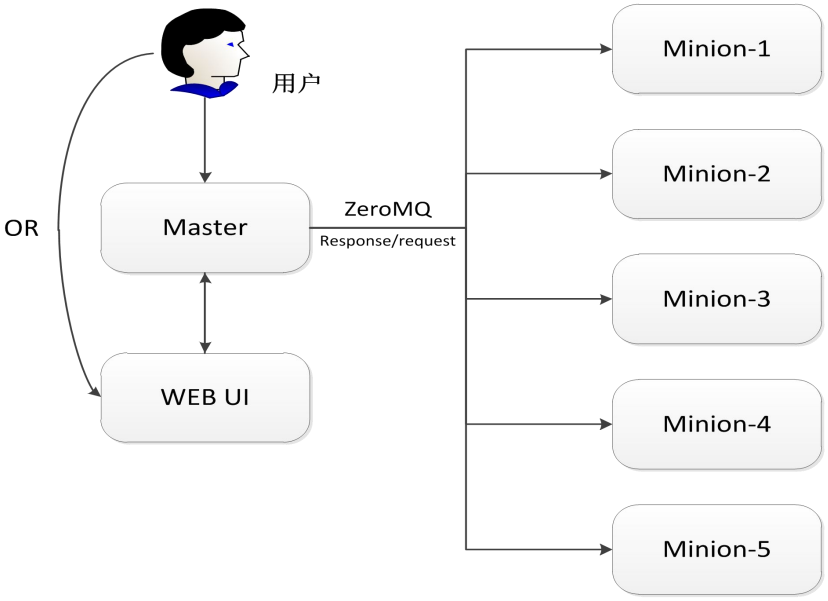

图2.4 ZeroMQ在SaltStack中的应用

2.4 高可用方案

2.4.1 双机热备

双机热备是一种常用的高可用方案，双机热备是将中心服务器通过一定的方法成为互为备份的两台服务器，并且在正常运行的情况下只能有一台服务器是保持Active，另外一台处于Standby[25]。如果正在运行的服务器出现故障导致无法正常启动时，另一台处于Standby的备份服务器会快速自动启动变为Active状态，双机之间的切换时间一般为2分钟左右，通过这种方式保证整个服务器系统的正常运行。双机热备的工作机制实际上就是为系统提供了一种故障自动恢复的能力，大大降低了故障恢复时间，从而保证系统的高可用性。双机热备可以借助集群工具实现，如PaceMaker，heardbeat等。

在本项目中，OpenStack系统和云数据库系统均采用了双机热备的形式，由于私有云平台和数据库服务通常需要长年累月的进行工作，因此备份工作是必不可少的，同时为了保证用户的使用高效性以及数据的安全性，所以采用的双机热备的形式保证系统高可用。

2.4.2 双机冷备

双机冷备份是相对于热备份来说的，与双机热备相同，冷备份也是对系统服务器进行备份成两台服务器，但是当故障发生时，双机冷备无法进行自动故障恢复，需要人为的进行服务器的切换，所以宕机时间过长，用于对系统使用频率不高的场景。双机冷备技术同样可以进行数据的定时自动同步功能，所以互为备份两台服务器之间基本不会发生数据大量丢失的情况。

2.4.3 集群工具

PaceMaker是一个集群资源管理器，它利用首选集群基础设施（OpenAIS 或heartbeat）提供的消息和成员能力，由辅助节点和系统进行故障检测和回收，实现性群集服务（亦称资源）的高可用性[26]。PaceMaker具有如下特点：

（1）主机和应用程序级别的故障检测和恢复；

（2）几乎支持任何冗余配置；

（3）支持多种集群的配置模式；

（4）支持定制应用启动/关闭顺序；

（5）支持STONITH确保数据的完整性；

（6）可以对不同的场景编写不同脚本进行集群管理；

（7）可以自动从其他节点复制并更新配置文件；

在云数据库服务器中，就是采用PaceMaker+Corosync的方案实现MySQL云数据库高可用性，将数据库启动所需要的服务用PaceMaker进行管理，当一个节点出现故障时，Corosync能及时检测到节点的异常，并通过PaceMaker自动启动备用节点中相应的服务，实现云数据库的高可用性。

2.5 本章小结

本章节简单介绍了本项目中所需要的技术背景，首先是介绍了Openstack，整个私有云平台就是基于Openstack进行设计的，因此对Openstack各个组件进行了分析，包括Nova，network，Cinder，Glance等重要组件，然后研究了云数据库服务器集群运维工具SaltStack，之后对RabbitMQ和ZeroMQ两种消息中间件进行介绍，并分析他们在Openstack和SaltStack中的作用；最后对几种高可用方案进行描述，包括双机热备、双机冷备以及高可用集群工具PaceMaker。

第三章 系统需求分析

本章根据市场中对数据库的实际使用情况，确定了云数据库的系统需求。云数据库是依托私有云平台实现的数据库的自动化部署。本系统旨在将数据库资源池化，用户可按需选择数据库类型，以及数据库磁盘大小、最大连接数、内存等，并通过Web Service 的形式向用户提供对实例的多种操作，包括创建删除、启动停止、备份恢复等功能，大幅简化用户操作数据库的复杂度。本系统还提供了对实例多项性能指标的监控，用户也可通过对数据库参数的修改进行数据库的性能调优。为了提高系统的稳定性和高可用性，底层数据库服务器采用了双机热备的形式，有效防止服务器宕机带来的危害。

3.1 数据库服务器管理

3.1.1 数据库管理两种存储方案

数据库服务器的存储方案直接决定了数据的安全可靠性，为了达到系统的高可用性，当主备服务器进行切换时，要保证数据的一致性，这样才能不影响实例的正常运行。在本项目的进展过程中，前后分别采用了DRBD与共享存储两种方案，以下分别介绍这两种方案。

（1）DRBD存储方案

DRBD（Distributed Replicated Block Device）是一个通过软件实现的、无共享的、服务器之间镜像设备内容的存储复制的解决方案。DRBD的每个资源都有个角色，Primary（主）或Secondary（备），主备角色可以互换，各角色的DRBD设备都接受来自对方的修改，但是应用程序读写是不允许的，当本地DRBD设备的数据发生更改，主备DRBD设备的数据将会进行同步，直到主备的数据保持了一致，如图3.1所示。

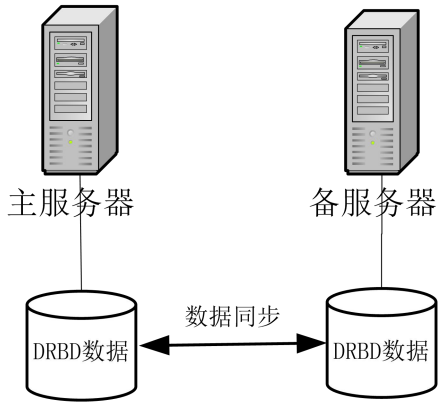

图3.1 DRBD数据同步

DRDB存储方案在用户没有共享存储且服务器存储容量较大时采用此方案较好，但缺点也很明显，由于需要数据同步，所以会占用比实际数据大小多一倍的容量，造成了资源浪费，第二点就是容易出现脑裂的情况，导致主从角色混乱。因此在项目后期采用了共享存储的方案。

（2）共享存储方案

共享存储指的是多台服务器对同一个存储设备的同一个分区进行访问。在本项目中采用的是H3C P5730存储服务器作为共享存储，在共享存储上划分出多个卷，挂载到数据库服务器上。在本方案中，由于采用的是双机热备，所以共享存储在某一个时刻只能挂载在主机或者备机其中一个上，同一时刻只允许一台服务器对共享存储进行读写。当其中主服务器宕机或者服务出现异常时，共享存储会在集群管理工具PaceMaker的帮助写自动切换到备服务器上，保证数据高可用性，如图3.2所示。

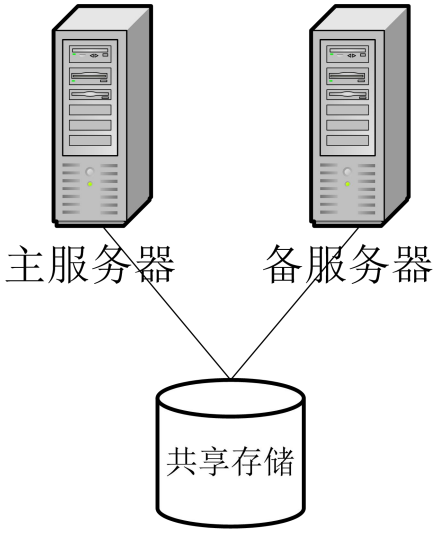

图3.2 共享存储结构图

3.1.2 添加数据库物理服务器池

数据库物理服务器是创建数据库实例所依托的物理环境，每种数据库类型对应一对物理服务器集群，即上文提到到双机热备高可用集群。为了满足云数据库的工作环境，需要在物理服务器中安装Pacemker、Corosync、MySQL、Percona等软件和工具。一对数据库物理服务器池对外提供一个虚IP供用户访问。用户在web页面点击新建物理服务器池，依次输入数据库类型，服务器IP地址，虚IP，组播地址，备份路径等信息，见表3.1，后台将会自动完成所选择数据库环境的自动化部署，最终提供给用户一对可用的物理服务器池，web页面见图3.3。

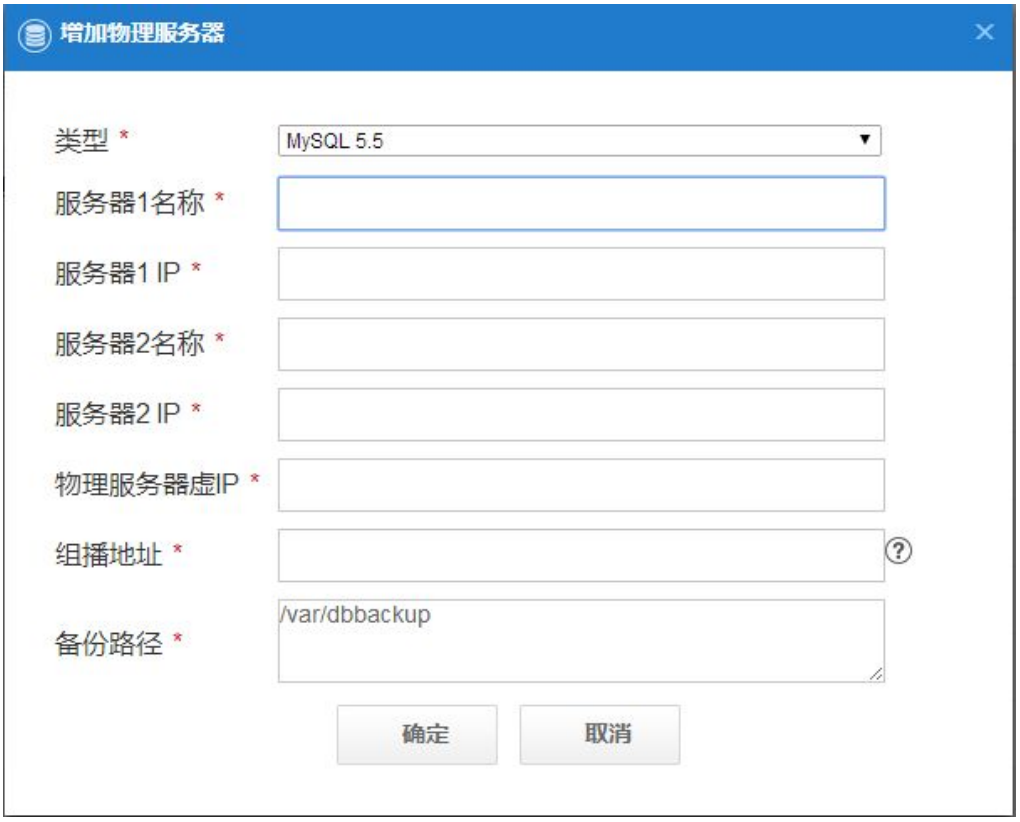

图3.3 添加物理服务器池界面

表3.1 添加物理服务器资源池选项用途

名称	用途
类型	选择所要创建实例的数据库类型，如MySQL5.5，MySQL5.6
服务器名称	为服务器制定名称，每个服务器名称唯一
服务器IP地址	物理服务器实际IP地址，添加资源池时以此IP地址进行通信
物理服务器虚IP	物理服务器池对外提供虚IP，用户通过虚IP访问数据库实例
组播地址	每对物理服务器拥有唯一组播地址，不同类型服务器池间通信混乱
备份路径	指定实例备份数据所存放的路径

当物理服务器创建成功后，系统还会将物理服务器的信息通过前台页面展示给用户，如图3.4。这些信息包括：服务器IP地址，主机状态，磁盘与内存的使用率，所创建实例数等信息。用户通过这些直观的信息可以判断目前服务器资源使用率，以便更合理的利用资源。当物理服务器无用时，用户同样可以通过页面对服务池进行删除。

名称	IP地址	虚IP地址	类型	主机状态	磁盘使用率	内存使用率	实例数	状态
mysql11	172.5.8.20	172.5.8.245	MySQL 5.6	可用	68%	39%	1	运行中
mysql22	172.5.8.21	172.5.8.245	MySQL 5.6	可用	68%	24%	1	运行中
nnn	172.5.8.77	172.5.8.243	MySQL 5.5	可用	50%	47%	0	运行中
mmm	172.5.8.78	172.5.8.243	MySQL 5.5	可用	50%	42%	0	运行中

图3.4 添加后的服务器状态

3.2 数据库实例管理

3.2.1 创建删除数据库实例

数据库实例是由数据库后台进程/线程以及一个共享内存区组成，共享内存可以被运行的后台进程/线程所共享，只有数据库实例才可以用来操作数据库文件。在MySQL中，实例和数据库的关系通常是一一对应的，即一个实例对应一个数据库，一个数据库对应一个实例，因此数据库实例对用户是必不可少的[27]。

在有可用的MySQL物理资源池的情况下，用户通过点击Web页面提供的新建实例按钮，并输入创建实例所需信息，见表3.2，图3.5，系统会自动去选择合适的服务器去创建数据库实例，并将实例信息返回到前台页面，用户可以通过页面中提供的虚IP地址，端口号，以及用户名和密码，远程登录到数据库实例中进行相关操作。也可以对无用的实例进行删除，后台会将此实例的数据从服务器里删除以节省资源。

图3.5 创建实例页面

表3.2 创建实例各选项用途

名称	用途
类型	选择所要创建实例的数据库类型，如MySQL5.5，MySQL5.6
实例名称	为实例创建名称，实例名称唯一
私有网络	关联私有网络，可以将云数据库与同一网段云主机连接
最大连接数	决定此实例的最大并发数，限制连接到此实例的用户数
容量	实例的存储数据的最大容量

3.2.2 云数据库实例的操作

在日常使用中，用户可能遇到有暂时不用的实例需要关闭以节省服务器的资源，也会遇到当前数据库配置无法满足工作需求的情况，针对这些问题本项目提供了对状态为可用的实

例提供了如下操作：

（1）实例停止：对于暂时不用的实例用户可以选择停止实例，实例停止后会将状态置为不可用，同时在后台服务器中也会将此实例进程停止；

（2）实例启动：对已停止实例，重新将服务器中的实例进程启动，启动后实例正常使用；

（3）修改：用户可以修改实例的名称以及描述，方便用户了解实例用途，以便后续工作中使用，见图3.6；

（4）实例升级：实例的升级包括最大连接并发数和实例容量的升级，增加最大连接并发数可以允许更多的用户同时连接到此实例上，增加实例容量可以存储更多的数据，见图3.7；

图3.6 修改数据库

图3.7 升级云数据库

3.2.3 数据库实例参数修改

随着数据库在各个领域的使用不断增长，越来越多的应用对数据库提出了高性能的需求。数据库性能调优是知识密集型的学科，需要综合考虑各种复杂的因素，如数据库缓冲区的大小，参数值是否合理，索引的创建，语句改写等[28]。总而言之，为了能让系统运行的更快，数据库的性能调优愈发的重要。

为了符合用户的需求，本项目提供了数据库性能调优的一种方式，即数据库实例参数修改。前台Wed页面为用户提供了几十个MySQL参数供用户修改，对于每个参数都给出了参数详细说明，见图3.8，其中参数名称、系统默认值、允许修改的范围值、参数的用途以及修改后实例是否需要重启，最大限度的保证了修改参数后实例的安全性和可用性。用户无需进入查阅相关参数的意义，也无需进入实例内部使用命令修改参数，只需在页面点击相关参数即可，极大的降低了操作复杂度，提高了工作的效率。

图3.8 其中一个参数详细的说明

3.2.4 备份恢复数据库

数据库在发生意外停机或者数据丢失时往往会造成非常严重的损失，为了保证数据库数据的高可用性，需要指定消息的数据库备份与灾难恢复的策略[29]。数据库的备份是一个长期的过程，但恢复只是在有事故发生后进行的，所以可以看成是备份的逆过程，而数据库恢复程度的好坏在很大程度上依赖于备份的策略和情况的好坏，云数据库为用户提供了全量备份，增量备份，以及备份策略的指定。

（1）全量备份：这种备份是大多人常用的备份方式，它可以将整个数据库进行备份，包括用户表、系统表、索引、视图和存储过程等所有的数据库对象[30]。但是缺点也很明显，全量备份需要花费大量的时间和空间。

（2）增量备份：这种备份是指在一次全量备份或者上一次增量备份后，以后每次的备份都只需备份与上一次备份相比增加的或者被修改的那部分文件。由于仅仅是备份增加或修改的文件，所以增量备份的数据量不大，备份所需要的时间也很短，但是增量备份恢复比较麻烦，它需要沿着最近一次全量备份到增量备份的时间顺序依次恢复，因此较大的延长了恢复的时间。

（3）备份策略：制定备份策略即是制定数据库备份的时间和方式，根据需要确定需要备份的内容、备份的时间及备份的方式。好的备份策略可以极大的提高数据的安全性，以较少的资源提供可靠的备份与恢复。

本项目的云数据库为用户提供了全量备份，即立即备份，并在增量备份的基础上制定了备份策略。备份策略的备份周期为七天，用户可以选择周一到周日的某一天或者某几天对数据库进行备份，可选的备份种类分为增量备份和全量备份，同时也提供的备份时间的选择，用户可选择一天二十四小时中的任意整点进行数据库备份，如图3.9。 以这些功能为基础用户可根据实际需求制定备份策略，但是七天中一定要有全量备份作为增量备份的基础，对于

备份策略中的自动备份系统会为其保留十五天，十五天后将会按照一定的规则进行删除，具体规则将在第五章中进行介绍。

图3.9 设置备份策略

3.2.5 实例性能监控与告警阈值

（1）性能监控

用户在使用数据库实例的过程中，需要对数据库实例的健康状况有一个直观的判断，了解实例的性能状况，这样可以有效的避免实例性能方面的问题。因此，云数据库为用户提供了性能监控的功能，通过对一些重要的性能指标参数的采集，记录数据库实例的实时信息和历史数据信息，并且以图表的形式动态的绘制出性能曲线图，实例监控的时间范围可以是一天、一周、一月、一年，如图3.10。

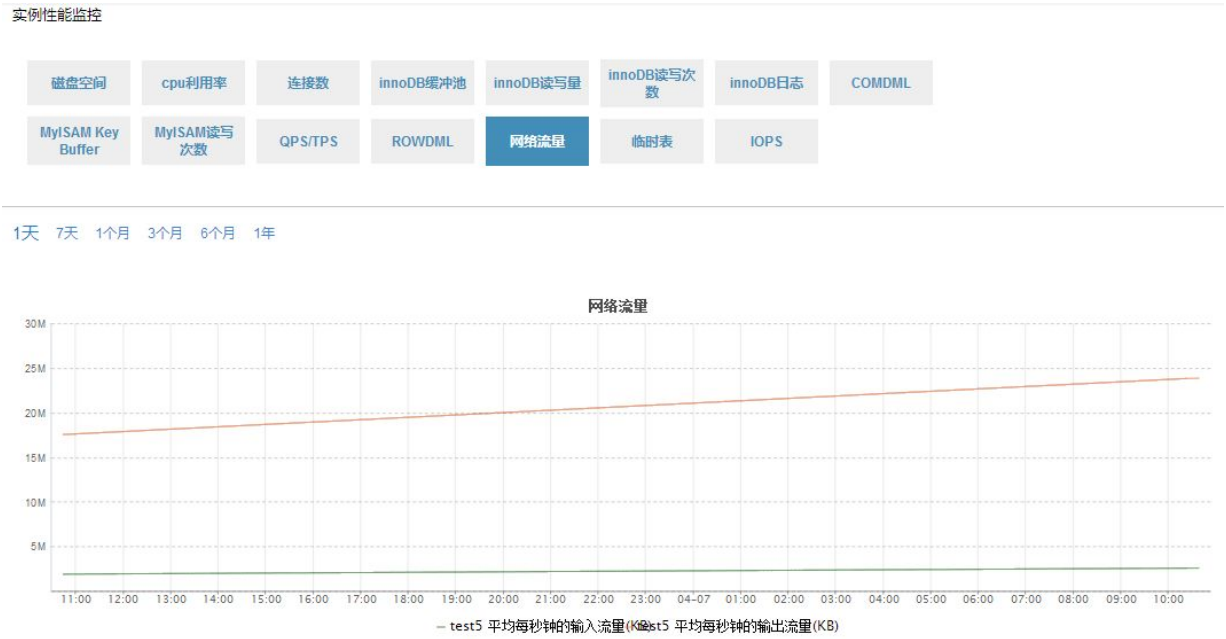

图3.10 性能监控图

性能监控中部分参数意义见表3.3。

表3.3：性能监控参数意义

监控项	含义
磁盘空间	实例的磁盘空间占用历史趋势，单位GB
CPU利用率	实例CPU的使用率（占操作系统的总数）
连接数	实例当前的总连接数
InnoDB缓冲池	InnoDB缓冲池的读命中率、利用率以及缓冲池脏块的百分比
InnoDB读写量	InnoDB平均每秒读取和写入的数据量
InnoDB读写次数	InnoDB平均每秒读取和写入的次数
InnoDB日志	InnoDB的日志写入情况
COMDML	数据库每秒语句执行次数的监控，所涉及到的语句包括Insert、Delete、Insert_Select、Replace、Replace_Select、Select六种
MylSAM Key Buffer	MylSAM平均每秒的Key Buffer使用状况
MylSAM读写次数	MylSAM平均每秒的读写次数
QPS/TPS	每秒钟SQL语句执行次数和事务处理数
ROWDML	InnoDB每秒钟操作数据行数的统计，根据操作的不同，分为平均每秒向日志文件的物理写次数、平均每秒从InnoDB表“删除/更新/读取

	/char”的行数等五种类型
网络流量	MySQL实例平均每秒钟的输入、输出流量，单位为KB
临时表	执行语句时在硬盘上自动创建的临时表的数量
IOPS	实例的IOPS（每秒IO请求次数）

（2）告警阈值

当数据量较大时，数据库实例在运行过程中会遇到实例性能不足或者物理资源不足的情况，这时有可能会造成数据丢失或实例异常的问题。云数据库为了防止这类问题提供了告警阈值的设置，用户对重要的性能资源设置告警阈值，如图3.11，当资源使用率达到设置的阈值时会发出警告。

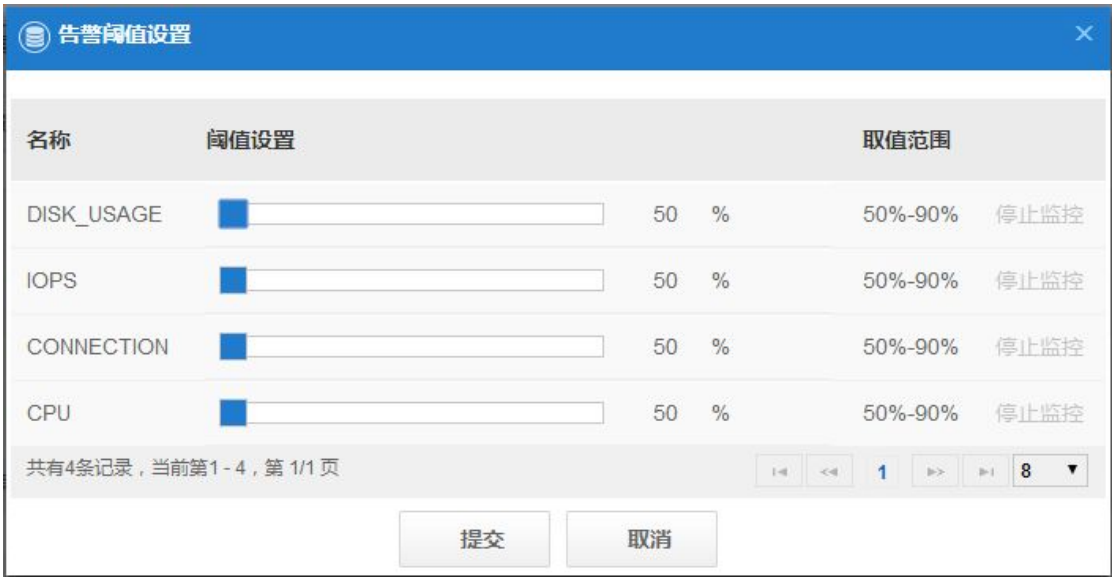

图3.11 告警阈值

3.2.6 临时实例

临时实例是基于备份集或七天内的一个时间点的备份恢复出来的一个临时的云数据库实例，该数据库实例最多能都保留48个小时，如果超过这个时间段将会被自动销毁。添加临时实例的功能是为了让用户能够临时查看历史数据。临时实例创建成功后，账号，数据库将继承备份集的数据。

3.3 数据库和用户

3.3.1 概述

数据库是指物理操作系统的文件或其他形式文件类型的集合，在MySQL中的数据库文件可以是frm、myd、myi、ibd结尾的文件。数据库定义在实例中，一般情况下，一个数据库对应于一个实例，一个实例可以包含多个数据库[31]，在同一个实例中的不同的数据库时完全独立的，它们分别拥有自己独立的系统编目表，比如用户可以在同一实例下创建不同的数据库区分不同的业务，不同业务的数据互相没有干涉。

数据库用户是具有登陆数据库并管理数据库权限的角色，在数据库实例创建完成后会生成一个超级用户名（MySQL一般为root），超级用户名具有最高权限，可以创建其他用户，并给用户赋予权限。每个数据库用户只能登陆到分配给它的数据库中，并可以在这些数据库里执行权限范围内的操作，见图3.12。通过为不同数据库用户指定不同的数据库，可以对不同用户的数据库进行隔离，提高数据的安全性。

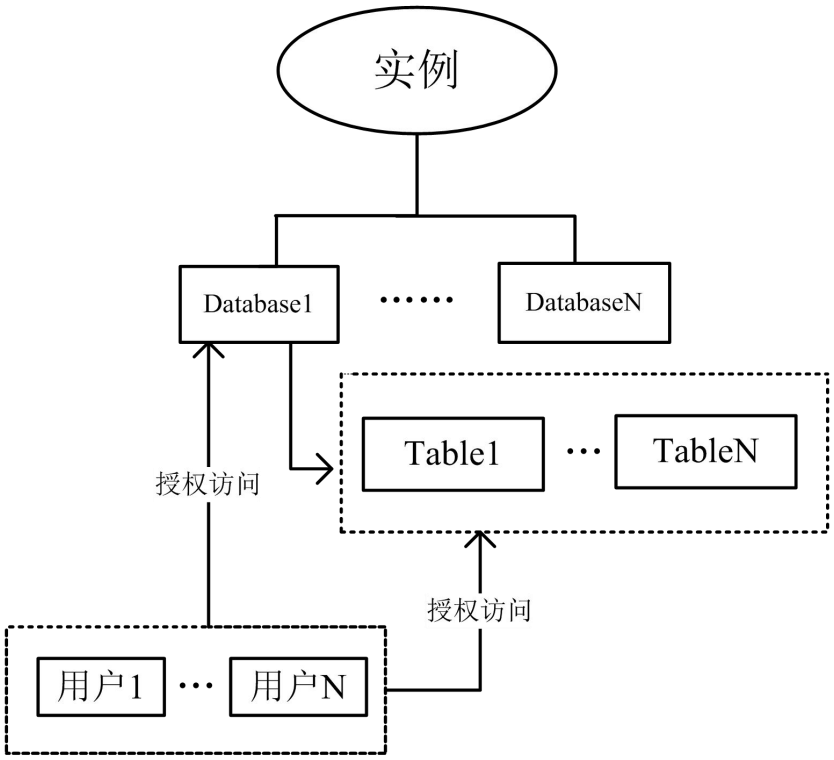

图3.12 MySQL数据库和数据库名关系图

3.3.2 数据库和用户管理

（1）新建删除数据库

用户点击页面新建按钮，输入名称和选择支持的字符集即可新建数据库，支持的字符集有utf-8、GBK、Latinl。新建后的数据库会与所在实例进行绑定，如需删除数据库直接点击删除，系统会自动从实例中删除此数据库。

（2）新建删除数据库用户

在已经新建好数据库的情况，选择新建数据库用户，按提示输入用户名和密码，这个用户名和密码是用来登陆数据库使用的。之后需要选择需要绑定的数据库与此用户的操作权限，操作权限有只读和读写两种，如图3.13。在新建好用户之后即可登陆操作数据库，操作权限和选择的读或写的权限时一致的。用户可以选择修改已有的数据库用户，修改选项包括密码和绑定数据库，如需删除用户直接点击删除，系统会在实例中自动注销此用户。

图3.13 创建数据库用户

3.4 本章小结

本章从系统需求的角度对云数据库的功能进行了介绍，首先是分析了云数据库存储架构，分别介绍了两种存储方案的，包括DRBD方案和共享存储方案，并分析对比了两种方案的优劣。

然后对添加物理服务器池的操作进行介绍，包括服务器池的创建删除等，之后是介绍数据库实例的操作，包括创建删除，启停，备份；最后对数据库及数据库用户的添加和工作方式进行了介绍。

第四章 系统体系结构

云数据库的实现是私有云平台的基础上，而私有云平台的项目又是基于OpenStack开发，私有云平台有包括云数据库的众多组件，如监控、告警等。为了实现资源的充分利用和系统的高可用性，私有云平台支持分布式部署。云数据库和OpenStack都才采用了高可用的系统架构方案，所以系统整体物理架构的设计需要考虑多方面的因素。云数据库项目中前台开发依赖的是Java、JavaScript等技术，由于这些技术不适合直接与底层设备进行交互，所以项目后台采用C++设计，底层数据库服务采用Python调用SaltStack完成对数据库集群的远程控制部署，在软件逻辑架构设计时既要考虑到前后台通信，又要考虑到底层设备的交互以及各种功能需求的合理实现。

4.1 OpenStack架构方案

4.1.1 物理架构

在实际的部署中，OpenStack可以根据不同的安装组件把服务器划分为计算结点（Compute Node）、存储节点（Storage Node）、网络节点（Network Node）和控制节点（Controller Node），如图4.1。在本项目私有云平台的部署中，控制节点采用的双机热备方案，保证控制节点的高可用性；计算节点采用集群设计，当处理数据时，Schedule调度器会选择最优的计算节点处理数据，从而提高数据处理能力。实际项目中可以根据需求决定计算节点的数量。

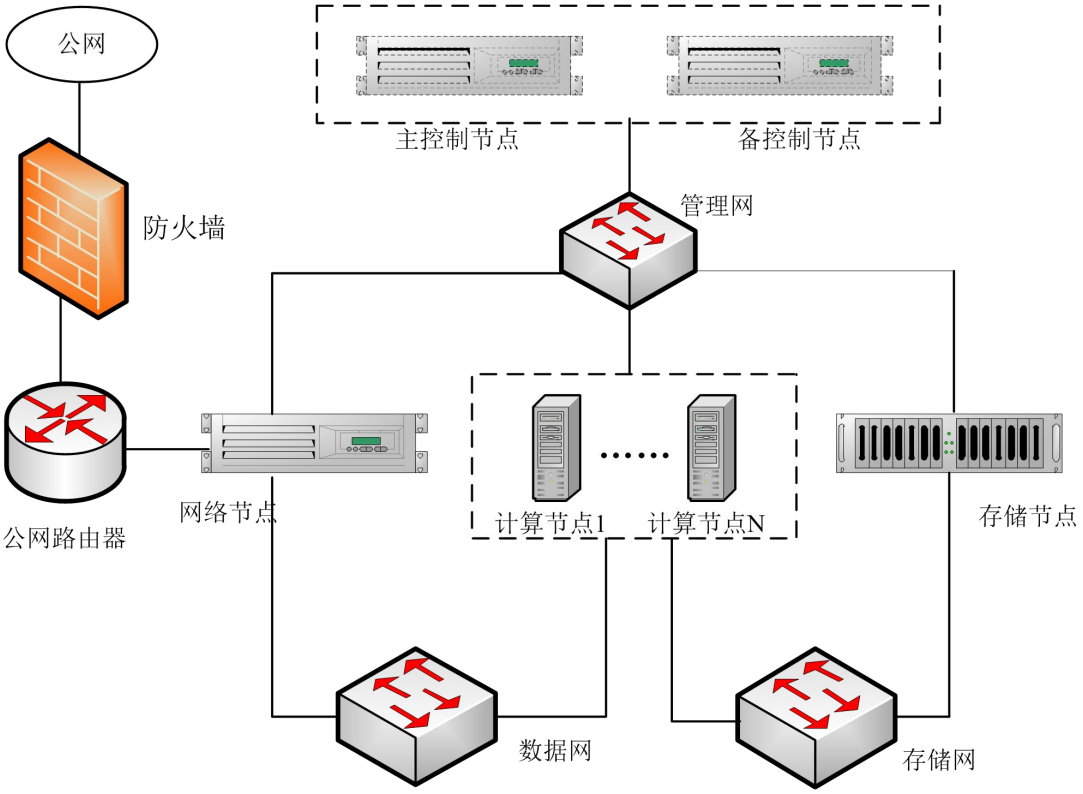

图4.1 OpenStack物理架构

控制节点中安装了Nova-api、Nova-schedule、keystone、RabbitMQ、network和MySQL数据库等有关Nova组件的服务器，在实现企业级应用的项目中控制节点都是由多台服务器共同组成，本项目中使用的主备两台服务器。

计算节点一般情况下是指安装了Nova-compute服务的物理机，在本项目中的计算节点主要包含Nova-compute、虚拟化插件等，计算节点主要是为虚拟机提供CPU、虚拟化内存等，本项目中计算节点采用的是集群环境，提高计算处理能力和高可用性。

存储节点一般情况下指安装有Nova-volume或Nova-Cinder的服务器，主要是为OpenStack系统提供块存储的服务，块存储和虚拟机的弹性云存储类似，Nova-Cinder可以从存储节点上为虚拟机动态实现增加或调整所需硬盘大小的功能。本项目中存储节点为共享存储，主要用于存放Glance镜像，Nova-Cinder与计算节点装在了一起，也可根据从共享存储中划取所需大小的磁盘挂载到计算节点下供Nova-Cinder使用。

网络节点一般情况下指安装有Nova-network/Neutron-server服务组件的服务器，主要功能是为OpenStack系统提供网络服务，包括DHCP server、虚拟网桥以及虚拟路由等功能。本项目中网络节点安装了Neutron-server，可以为虚拟机实例创建网络端口，虚拟网络，子网等功能。

管理网，数据网和存储网是根据交换机组网区所属的业务为其命名的，他们的功能就是

保证每个节点间的网络畅通，数据信息可以正常通信。防火墙添加有相应的规则集，可以保证业务网的安全运行。

4.1.2 逻辑架构

OpenStack Compute是OpenStack的核心组成部分，Compute负责计算服务的相关操作，这里主要介绍OpenStack Compute的逻辑架构[41]。在Compute的逻辑架构中，绝大部分的组件可以分为两种自定义编写的Python守护进程：接收和协调API调用的WSGI应用（Nova-api，Glance-api，etc）、执行部署任务的Worker守护进程（Nova-compute, Nova-network, Nova-schedule, etc.）[32]。消息队列和数据库即不是自定义编写，也不是基于Python，而是简化了通过消息传递和信息共享任务的异步部署。逻辑架构如图4.2。

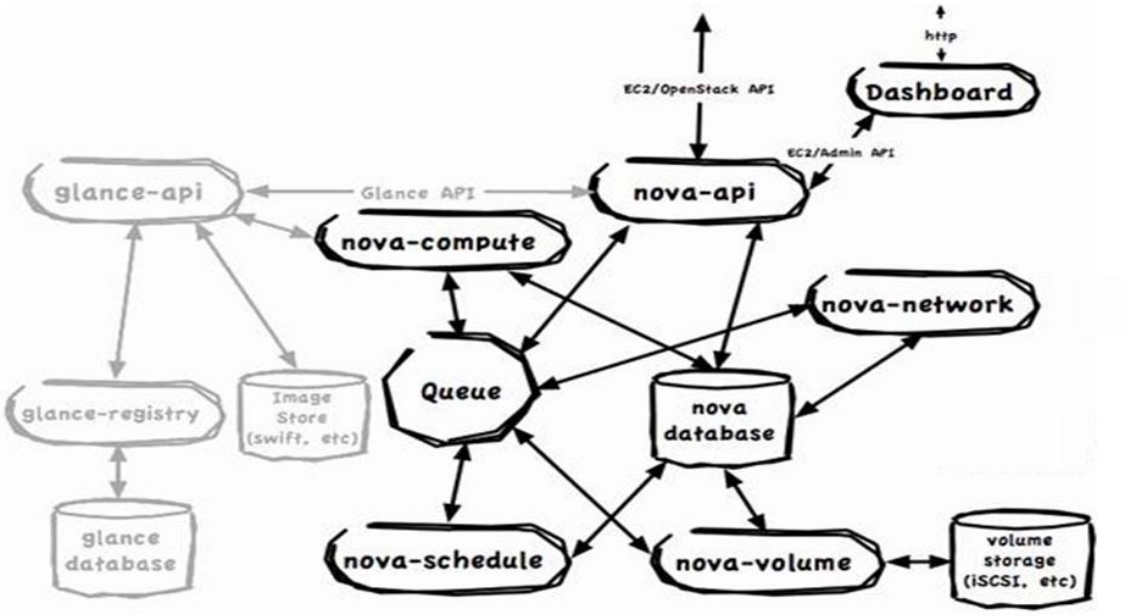

图4.2 OpenStack Compute逻辑架构

从上面的架构图中，我们可以得到出三点[33]：

（1）终端用户（DevOps, Developers 和其他的 OpenStack 组件）通过和 Nova-api 对话来与 OpenStack Compute 交互。

（2）OpenStack Compute 守护进程之间通过消息队列（行为）和 database（信息）来交换信息，以执行来自的 API 请求。

（3）OpenStack Glance 基本上是独立的基础架构，OpenStack Compute 通过 Glance API 来和它交互。

4.2 私有云平台架构方案

4.2.1 物理架构

私有云平台是基于OpenStack平台的云服务平台，在继承原有架构灵活、扩展性强、开放性和兼容度高的基础上，提高了平台的稳定性和可靠性[34]。该平台是基于租户到应用的端到端的云服务配置和管理，将用户申请的服务组装成服务链，统一管理和配置。云平台支持的服务申请包括：云主机、云存储、网络、云LB、云防火墙、云数据库等。由于本论文主要介绍云数据库，所以只突出包含云数据库的私有云平台物理架构架构。仅包含云数据库的私有云平台结构主要包含：平台组件的部署、Openstack的服务部署、云数据库的部署，如图4.3。

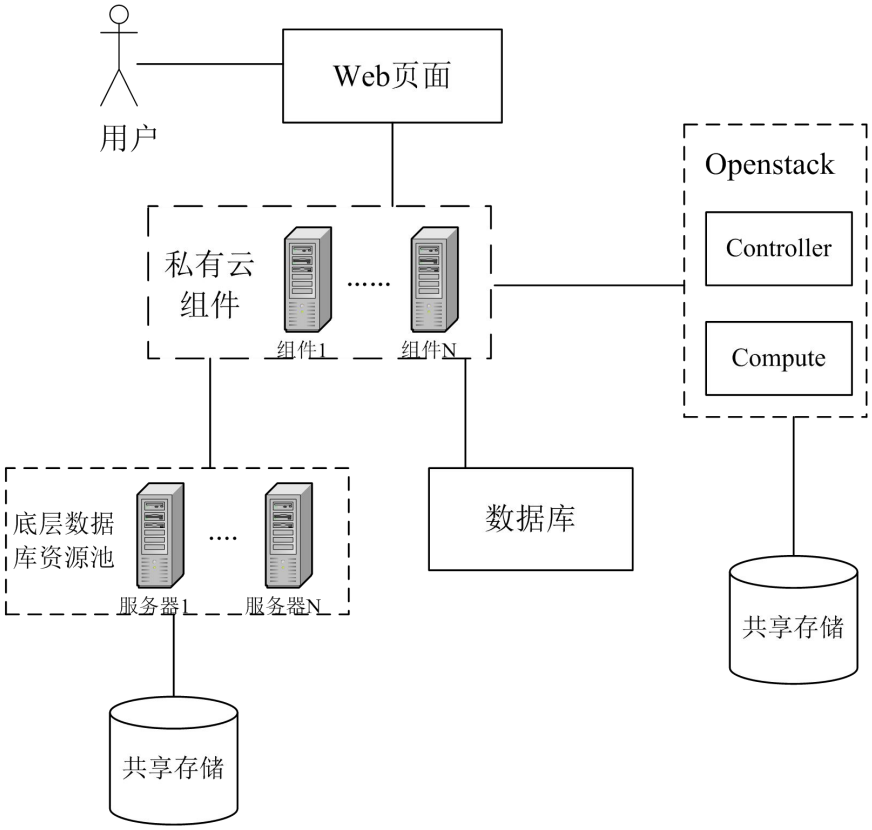

图4.3 私有云平台物理结构

私有云平台组件的部署分为集中式和分布式，集中式部署就是所有组件都装在一个服务器上，这样做的优点是可以节省物理服务器资源，但是由于是所有组件共享同一台服务器的资源，所以很有可能出现资源不足的情况，导致系统响应缓慢，第二个缺点是容灾性能差，一旦服务器出现故障所有组件都将瘫痪。分布式部署是将不同组件按需部署到不同的服务器上，组件可用资源充足响应速度快，提高了系统性能，分布式的环境也增加了系统容灾性能，实现了高可用。组件所连接数据库是用来存放各个组件运行所需的数据信息，已经web页面

中展示给用户所需要的数据。

OpenStack组件部署的详情情况已在上一节给出，OpenStack在平台部署中是为平台的工作提供API接口，平台各组件通过对OpenStack的API调用完成用户请求服务的响应，共享存储是作为Nova-Cinder块存储或者Glance镜像存储使用。

底层数据库资源池是云数据库部署环境，数据库资源池的部署也采用了集群式部署，本项目采用的是双机热备式集群，每两个服务器为一对。在云平台组件中会有云数据库的后台进程，数据库资源池为进程发出的请求提供所需的物理资源和响应。共享存储是用来存储用户所申请的云数据库资源，如第三章所述，项目中还采用了DRBD的存储环境，这里不再详细描述。

4.2.2 逻辑架构

由于私有云平台是基于OpenStack的云服务平台，所以私有云平台的逻辑架构主要是云平台调用OpenStack API接口在不同组件之间相互协工作工作的过程。在逻辑架构中主要分成三大块：私有云平台（包含OpenStack API）、OpenStack controller Node、Region Node，如图4.4。其中私有云平台和OpenStack controller Node之间可以是一对多的关系，同样OpenStack controller Node和Region Node间也可以是一对多的关系。

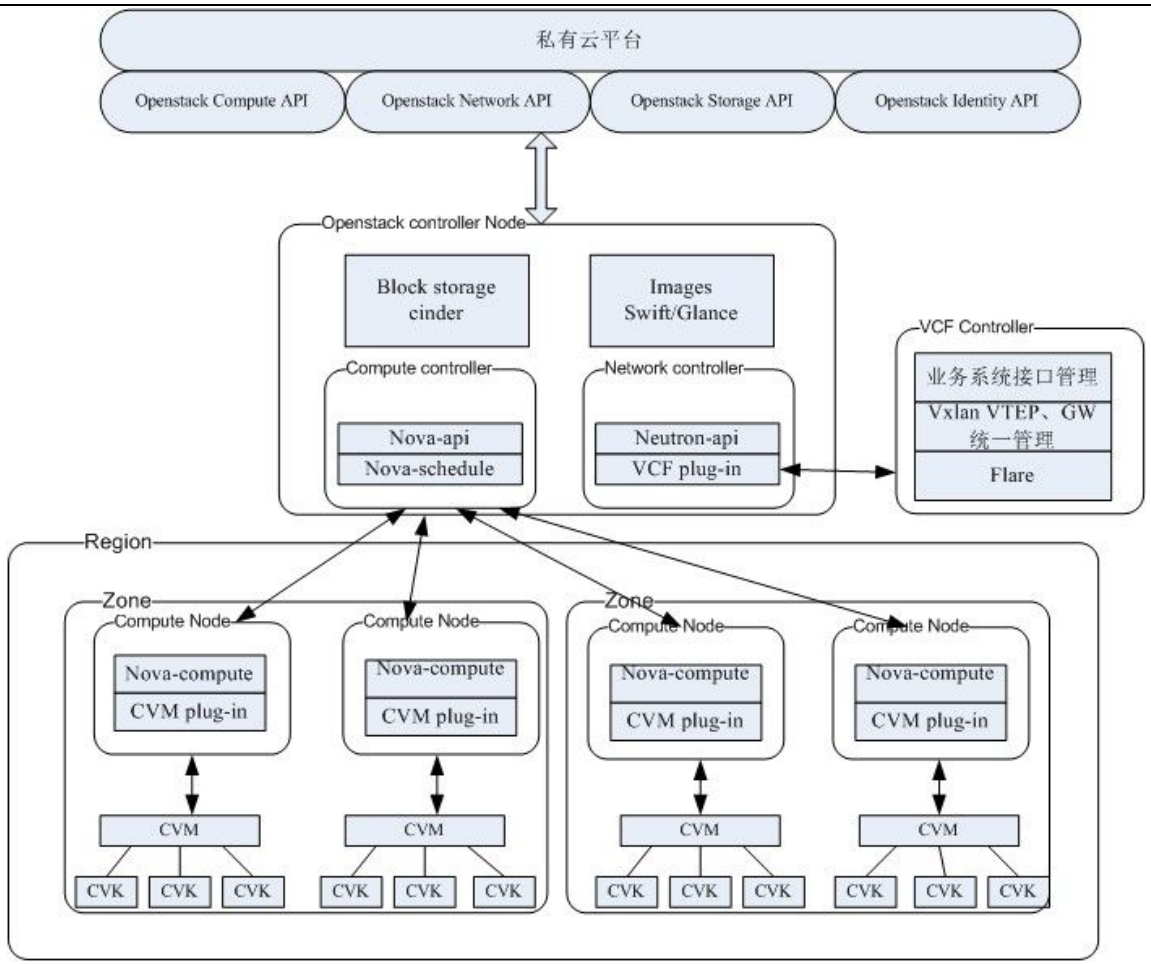

图4.4 私有云平台逻辑架构图

在本项目的私有云平台中除了用到了OpenStack的相关组件，还用用到了CVM虚拟化插件和技术，以及VCF虚拟化插件，下面分别简单介绍：

- CVM (Cloud Virtualization Manager，虚拟化管理系统)

主要实现对数据中心的计算，网络和存储等硬件资源的软件虚拟化管理，对上层应用提供自动化服务[35]。安装 CVM 后，可将计算、网络、存储进行虚拟化集中统一管理，并通过集群的高可靠性和动态资源调度功能、虚拟机的容灾与备份功能来确保数据中心业务的连续性。

- CVK（Cloud Virtualization Kernel，虚拟化内核平台）

运行在基础设施层和上层客户操作系统之间的虚拟化内核软件。针对上层客户操作系统对底层硬件资源的访问，CVK 用于屏蔽底层异构硬件之间的差异性，消除上层客户操作系统对硬件设备以及驱动的依赖[36]。

- VCF 控制器（Virtual Converged Framework）

VCF是一款SDN控制系统，SDN（Software Defined Network，软件定义网络）是一种新型网络创新架构，其核心思想是将网络设备的控制层面与转发层面分离，以实现对网络流量

的灵活控制，为网络及应用的创新提供良好的平台。VCF可以向上层云计算系统提供 API 接口和插件，方便云计算系统整合数据中心网络资源，实现“一站式”服务和管理。

私有云平台创建云虚拟机是云平台的一个典型功能，通云主机的创建可以清晰的了解到私有云平台架构中各组件之间的逻辑关系，下面创建云主机为例描述云平台的逻辑架构，如图4.5。

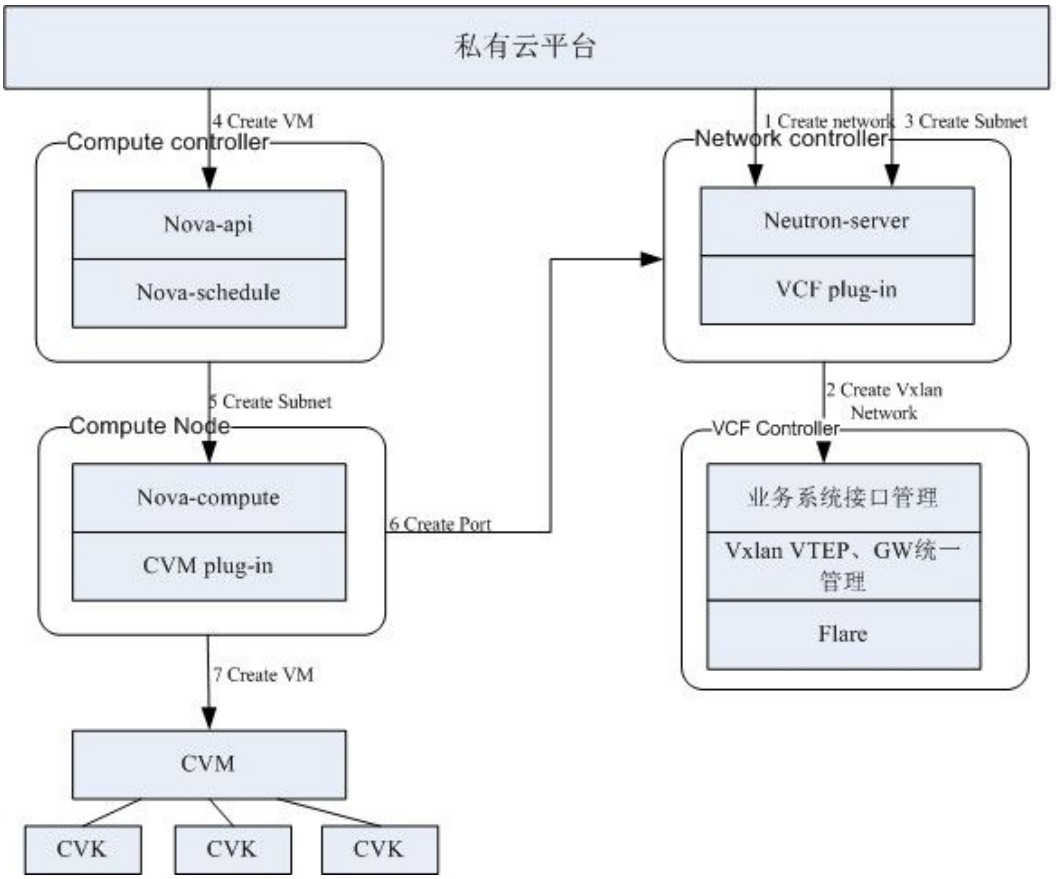

图4.5 私有云平台创建主机逻辑调用关系

用户在私有云平台Web UI选择创建云主机，具体组件间逻辑调用关系见表4.1：

表4.1 创建云主机逻辑调用

步骤	组件	动作
1	Neutron Controller	云平台调用Neutron Controller的Rest API创建一个网络，一个网络对应于一个通过VLAN或GRE或VXLAN隔离的网络区域。
2	VCF Controller	Neutron Controller通过内嵌的VCF plug-in，调用VCF Controller分配一个VXLAN网络隔离区域。
3	Neutron Controller	云平台调用Neutron的Rest API为步骤1中的网络创建一个子网，一个子网包含一个IP地址池，用于为网络中的虚拟机分配IP地址。
4	Nova Controller	云平台调用Nova Controller的Rest API创建一台VM，创建VM时选择的网络为步骤1中的网络。Nova Controller根据各个Nova Compute的负载信息，选择一台负载最小的Nova

		Compute。
5	Nova Compute	Nova Controller调用Nova Computer的Rest API创建一台VM。Nova Compute调用Neutron Controller的Rest API为VM创建一个端口
6	Neutron Controller	Neutron Controller分配一个MAC地址，从网络的子网中分配一个未使用的IP地址，返回给Nova Compute。
7	CVM	Nova Compute通过内嵌的CVM plug-in，调用CVM创建一台VM。

4.3 SaltStack架构方案

4.3.1 物理架构

SaltStack最大的特点就是对集群提供了极其灵活的配置管理和远程执行[37]。配置管理包括配置的下发，版本更新，制定依赖关系和状态管理等。远程执行是指下发命令倒Minion机器上执行。因此在实际部署中往往将Salt-Master与Salt-Minion分离开，把Salt-Master单独部署到一台机器上，或者对Salt-Master采用高可用架构，本项目中采用的是单独部署的方式，通过Salt-Master对集群下发各种命令，如cmd，MySQL，package等。逻辑架构如图4.6。

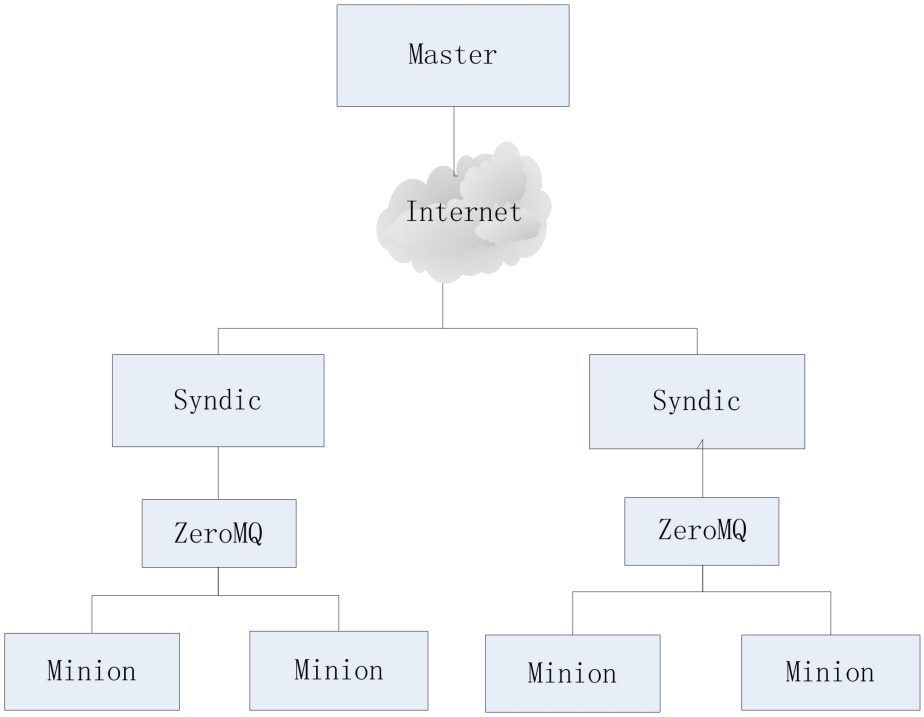

图4.6 SaltStack物理架构图

部署上，SaltStack分为Master。Syndic，Minion三部分，下面分别介绍这三部分：

（1）Master：即安装有Salt-Master的节点，它也可以看成SaltStack的控制节点，主要负责存储SaltStack的配置信息、对可信的Minion节点进行授权管理，通过消息中间件ZeroMQ与

各个Minion节点进行交互。

（2）Minion：即安装有Salt-Minion的节点，Salt-Minion就是一个运行在集群服务器上的agent进程，通过消息中间件ZeroMQ与Master进行交互，接收来自Master的命令，执行并返回结果。

（3）Syndic：一般在规模较大的部署环境中才会使用到syndic，比如在多个数据中心的部署环境中。syndic充当一个正向代理节点的角色，它代理了所有Master节点与Minion节点的之间的通信。通过这种方式可以将Master的负载分担给多个syndic处理。另一方面，它也可以降低Master通过广域网访问Minion的成本，具有更强的安全性，使SaltStack适用于大规模的数据中心的部署。

4.3.2 逻辑架构

在云数据库项目中，SaltStack负责对数据库集群服务器的运维控制，除了数据库相关的远程命令外，SaltStack本身也需要各种状态监测和命令执行，这就需要各模块间的紧密配合。SaltStack的逻辑架构如图4.7.

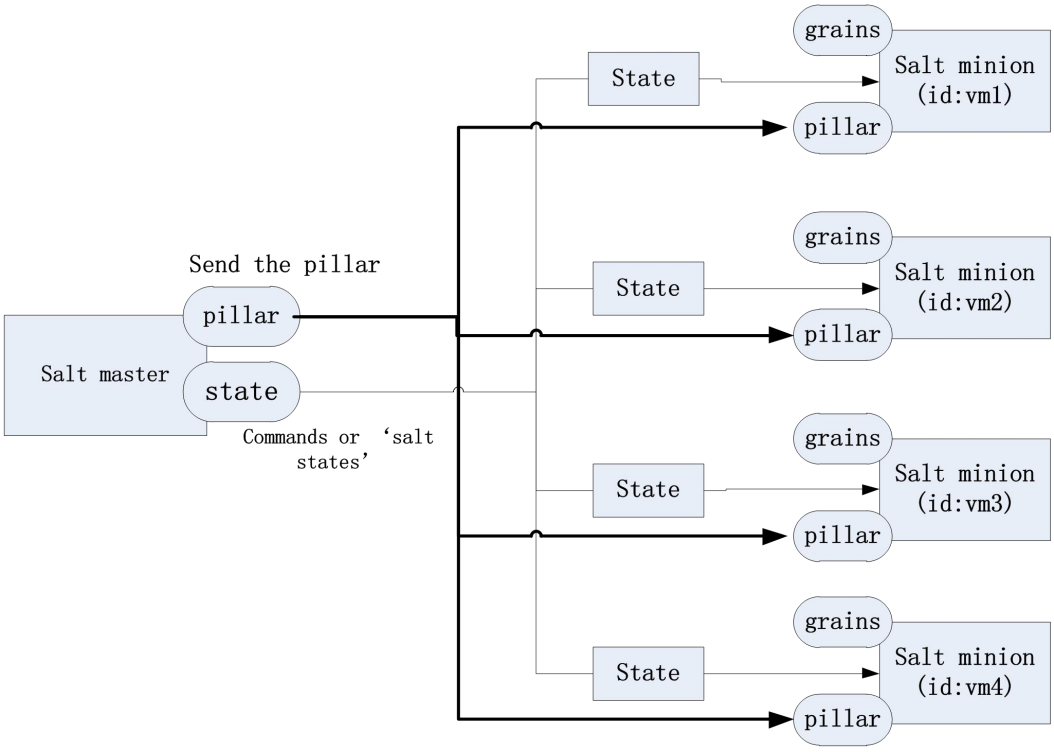

图4.7 SaltStack逻辑架构图

各模块的作用见表4.2：

表4.2 SaltStack各模块作用

名称	功能
Salt-Master	控制中心，Salt命令运行和资源状态管理端
Salt-Minions	装在被操作的服务器上
grains	Minion端的变量，属于静态变量
pillar	Minion端的变量，动态的，比较私密的变量，可以通过配置文件实现不同Minions自定义
state	配置管理的指令集

在Minion第一次启动的时候会采集静态数据，这个静态数据就是grains，它是以Python字典形式存放在Minion端，可以用在SaltStack的模块和其他组件中。事实上grains在每次Minion启动或者重启的时候都会进行静态数据的采集，即向Master汇报一次，因此grains没有pillar的灵活性强，比较适合做一些静态的属性值的采集的工作，如系统信息，磁盘个数，设备名称等比较固定的属性。

在SaltStack中，如果需要向Minion传递敏感数据，如ssh key，加密证书等，就需要用到pillar，pillar使用的是独立加密的session，pillar在解析完成后，是一个嵌套的dict结构，最上层的key是Minion Id，其value是该Minion所拥有的pillar数据，这些key-value数据存放在Master端，由Master编译好，Minion启动时会连接Master获取最新的pillar数据。

States是SaltStack的拓展模块，主系统使用的状态系统叫SLS系统，Salt状态是一些文件，其中包含有关如何配置Salt子节点的信息。当希望有SaltStack执行一套操作时，比如安装vim。配置vim，这样可以写在state里，也可以拆分成两个state写，执行时，Master将state生成好，下发给Minion，由Minion将其转换成一条条的命令，执行states。

4.4 云数据库架构方案

4.4.1 物理架构

云数据库底层架构设计时采用了一主多从的设计方式[38]，即一个主控制服务器控制多个从服务器，主控制器从云数据库组件进程中接收请求信息，再远程控制从服务器完成相应的响应操作。在云数据库集群设计中采用了系统自动化配置和管理工具SaltStack，在主服务器中安装Salt-Master，从服务器中安装Salt-Minion，如图4.8。

物理服务器池中的从服务器采用双机热备的高可用架构，采用集群管理工具PaceMaker

和Corosync的方案，正常情况下，主服务器处于online状态备服务器处于standby状态，这时的共享存储是挂载在主服务器上，当Corosync检测到主服务器异常宕机，会自动将备服务器切换为online状态，此时PaceMaker会将共享存储切换至备服务器上，并启动相关数据库服务保证数据库的可用性。每对主备服务器对外提供一个虚IP，用户通过虚IP登陆指服务器中的数据库里进行创建实例等操作，虚IP的切换同样受到PaceMaker和Corosync的控制，虚IP服务始终启动在online状态的服务器上。

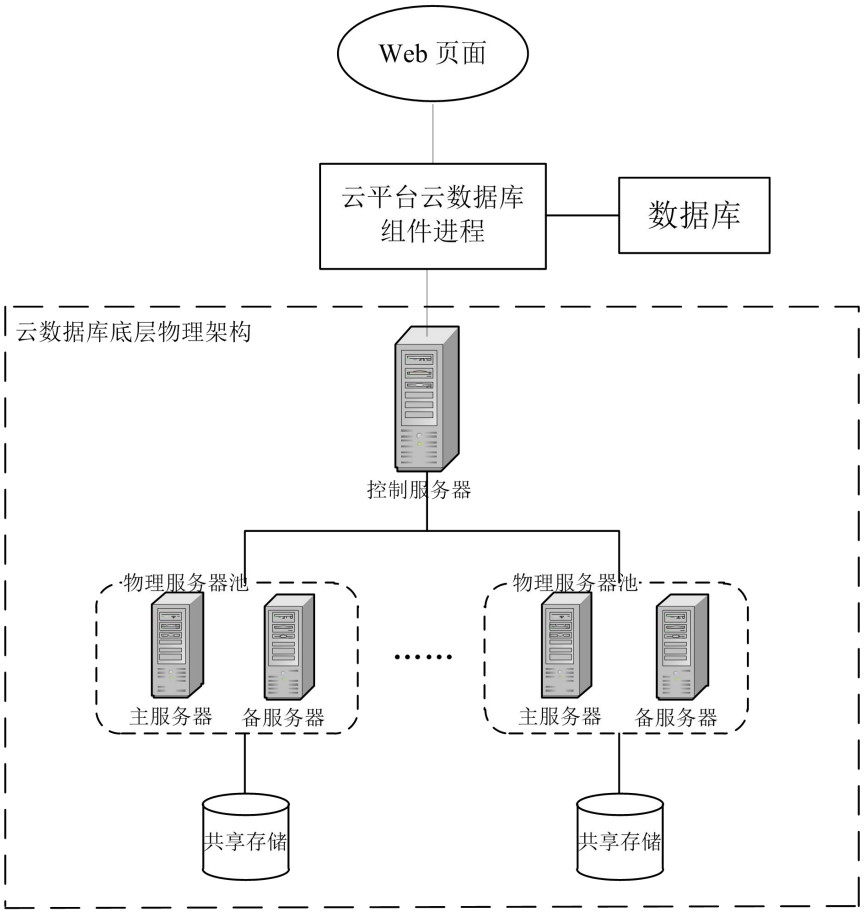

图4.8 云数据库物理架构

4.4.2 逻辑架构

（1）消息格式

（a）ASN.1

ASN.1即抽象语法符号，用来定义应用程序数据和表示协议数据单元的抽象语言。优点是独立于机器、语言及应用程序的内部表示[39]。适用于描述现代通信中复杂的、变化的、可拓展的的数据结构。在本项目中ASN.1用来描述前后台的通信中需要的数据结构，所用的语言分别是java和C++。

（b）Protocol Buffer

Protocol Buffer是google的一个开源项目，它适用于结构化数据串行化的灵活、高效、自动的方法，例如XML，不过它比XML更小更快，也更简单。使用者可以定义自己的数据结构并用代码生成器来读写这个数据结构，并且可以在无需重新部署程序的情况下更新数据结构。本项目中采用Protocol Buffer来描述后台C++和底层Python之间的数据结构。

（2）通信逻辑

云数据库的逻辑架构中主要有三个模块：数据库前台，数据库后台和底层服务器。模块间的通信要考虑消息格式，通信方式以及通信效率等各方面的问题。底层服务器采用的SaltStack管理，控制服务器和Minion之间通信依赖SaltStack的消息队列，如图4.9所示。

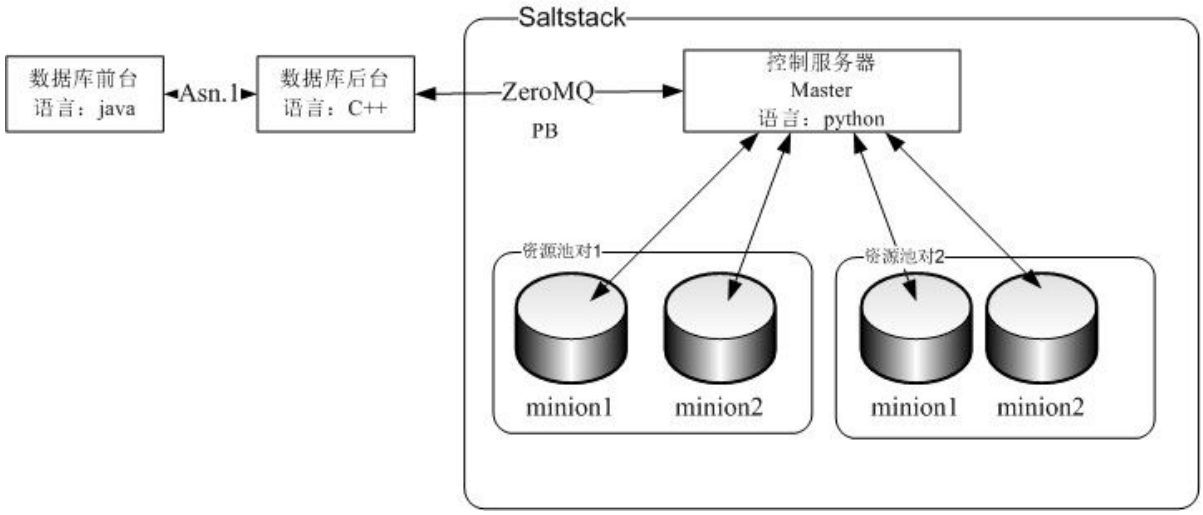

图4.9 云服务器逻辑架构

首先用户在Web UI选择相关功能会触发代码此功能的一个消息码，此消息码会放入OpenStack的RabbitMQ中，云平台的后台会从RabbitMQ中取得此消息码，并初始化相应的ASN.1数据结构，此时的ASN.1是用java语言描述的，经过序列化后发到C++端，C++解码后再根据内容初始化相应的ASN.1数据结构，经过C++后台处理后转换为Protocol Buffer数据机构格式，再序列化后经ZeroMQ发至底层Python，底层Python调用SaltStack的结构完成对Minion的操作，完成操作后获得相应的消息内容后再以Protocol Buffer返回后台，后台对云平台的数据库进行修改后把结果返回到页面展示给用户。

4.5 本章总结

本章主要设计并介绍了云数据库系统的组成架构，每部分都从物理架构和逻辑架构两方面进行分析。首先介绍了云数据库所依赖的私有云平台，由于私有云平台基于OpenStack开发

而成，因此又介绍了OpenStack的架构，然后是分析了云数据库部署中用到的SaltStack，最后遵循易用性和可靠性的原则，设计了云数据库的部署架构，并简要介绍模块间通信用到的消息格式。

第五章 系统功能详细设计与实现

在本文的第三章和第四章中分别制定了云数据库系统的功能需求和架构设计，本章在前两章的研究基础上，分析每个功能需求并给出具体的实现方案。每个功能需求都从功能设计、数据库表设计和关键技术实现三个方面去介绍需求的功能类设计，数据库表结构以及具体的实现流程。在进行功能类设计时，遵循高内聚，低耦合的原则，在设计数据库时遵循第三范式。

5.1 数据库物理资源池

5.1.1 功能设计

在需求分析时提出物理资源池是创建数据库实例的基础，为了遵循高内聚的原则，将与物理服务器相关的方法都包含类PhysicalServerTask中，其中包括物理服务器的添加，删除，刷新等操作。每一次的消息处理都会初始化一个task任务线程去处理相应的请求，PhysicalServerTask类的结构设计如下：

```
class PhysicalServerTask
{
public:
    ......
    void timerCallBack();
    static int refreshPhysicalServerInfoReq();
    static int dealRefreshPhysicalServerInfoResp();
    void refreshServerInfoReq();
    void addPhysicalServer();
    void deletePysicalServer();
    void addPhysicalServerPYSuccess();
    void addPhysicalServerPYFail();
    ......
}
```

PhysicalServerTask类中主要方法的含义在表5.1中给出：

表5.1 物理资源池相关方法

方法	作用
addPhysicalServer()	添加物理服务器
deletePysicalServer()	删除物理服务器
addPhysicalServerPYSuccess()	添加物理服务器池成功，修改相关数据库表
addPhysicalServerPYFail()	添加物理服务器池失败，修改相关数据库表
timerCallBack()	定时器
refreshPhysicalServerInfoReq()	请求刷新物理服务器池信息
dealRefreshPhysicalServerInfoResp()	处理刷新物理服务器池返回信息，修改相关数据库表

5.1.2 数据库表设计

数据库物理服务器池是创建云数据库的基础资源设施，在本项目中有MySQL、Oracle、SQL Server多种数据库，因此我们需要数据库表来记录物理服务器池的类型、虚IP、组播地址等信息确定创建数据库所需的服务器池。数据库物理服务器池的状态也需要呈现给用户，比如是否可用，磁盘利用率，内存利用率等信息。当创建数据库实例时，也需要综合各种服务器剩余资源的大小，选择最优的服务器池进行实例的创建。云数据库物理服务器池的相关信息保存在数据库表service-pool-list中，具体信息见图5.1。

service-pool-list	
PK	id
	name ip vip backup_path db_type disk_total memory_total status deleted broadcast_ip deploy_status disk_remained memory_remained

图5.1 表service-pool-list

数据库表service-pool-list中各个字段的详细信息如下表5.2给出。

表5.2 表service-pool-list字段意义

字段名称	数据类型	约束	备注
id	INTEGER	非空，唯一	列表主键
name	VARCHAR	非空	名称，长度256
vip	VARCHAR	非空	虚IP，长度256
backup_path	VARCHAR	无	备份路径，长度256
db_type	INTEGER	无	实例类型
disk_total	INTEGER	无	磁盘总计大小
memory_total	INTEGER	无	内存总计大小
status	INTEGER	无	数据库状态，是否可用（0：不可用，1:可用）
deleted	INTEGER	无	是否删除
broadcast_ip	VARCHAR	无	组播地址，长度256
deploy_status	INTEGER	无	部署状态
disk_remained	INTEGER	无	磁盘剩余大小
memory_remained	INTEGER	无	内存剩余大小

5.1.3 具体实现

在添加物理服务器池时，私有云平台会对用户在web页面输入的信息进行前端校验，校验的内容包括：IP格式是否正确、字符串的长度是否合理、组播地址的范围是否正确等信息，如果信息正确，前台会将这些消息以ASN.1的编码格式发送至C++端，如果校验失败，会给用户提示相应的信息进行更改。

C++端收到消息后进行ASN.1解码，然后将web页面输入的相关信息写入数据库表service-pool-list，此时服务器池状态为不可用（因为还在添加中），之后会ping服务器池的IP地址，如果ping不通，则输入的IP地址无效，无法创建服务器池，如果能ping通则将消息进行Protocol Buffer编码，并将消息通过消息中间件ZeroMQ发送到SaltStack中Master所在服务器，由于这部分由Python编写，我们称之为Python端（下面也会采取同样的叙述），这时的Protocol Buffer消息中包含的主要信息有：服务器池的数据库类型、服务器池IP地址、虚IP等信息。

Python端收到消息后首先进行Protocol Buffer解码，然后调用SaltStack的接口对消息中IP

地址所在的服务器进行集群配置，包括创建虚IP，安装PaceMaker、Corosync、MySQL数据库等，同时会在共享存储上创建VG（Volume Group）用来后续的实例创建，配置成功后在Salt-Minion上查看集群信息，会如图5.2所示。当检测到集群配置成功后，会刷新并获取服务器的资源信息，包括磁盘利用率，内存利用率，实例数等信息，将获取的信息同样以Protocol Buffer的编码格式返回至C++端，C++端会将Python端获取到的信息写入表server-pool-list对应的字段中，并将主机状态置为可用，至此，物理服务器池添加成功，上面有任何一个环节出错，都会直接改写数据库表中的服务器状态字段为添加失败。具体流程如图5.3所示。

```
Last updated: Thu Apr 16 05:35:29 2015
Last change: Wed Apr 15 14:01:39 2015 via crm_attribute on node1
Stack: corosync
Current DC: node1 (738527252) - partition with quorum
Version: 1.1.10-42f2063
2 Nodes configured
14 Resources configured


Online: [ node1 node2 ]

p_ip_mysql      (ocf::heartbeat:IPaddr2):       Started node2
p_fs_mysqlbackup        (ocf::heartbeat:Filesystem):    Started node2
```

图5.2 资源池添加成功

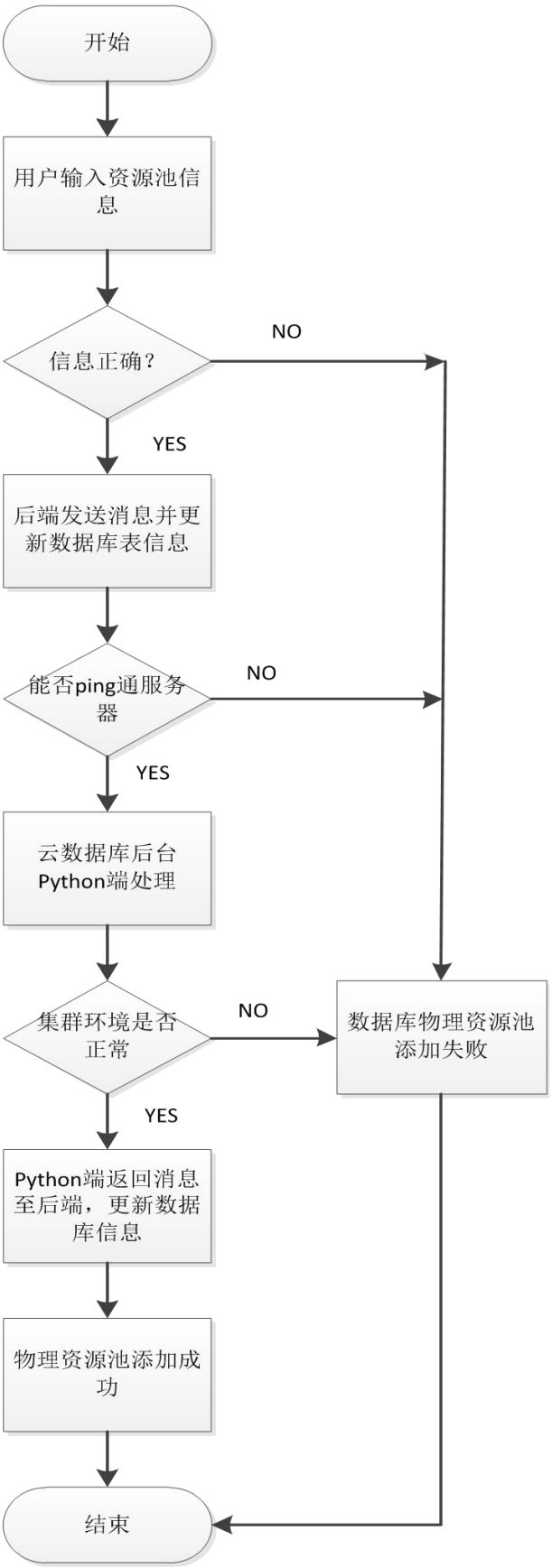

图5.3 添加物理资源池流程图

5.2 数据库实例创建与删除

5.2.1 功能设计

数据库实例的创建和删除是云数据库的核心功能，所涉及的方法包含在类DBEntityTask中，DBEntityTask类所包含的主要方法有创建删除数据库实例，选择最优服务器池等操作。创建实例采用的是多线程设计，每次创建实例都会起一个Task线程去处理，最高支持20个线程同时并发，类DBEntityTask具有设计如下：

```
class DBEntityTask
{
public:
    ......
    bool checkCreateStatus();
    bool static selectBestServer();
    static void pingServerGroupIP();
    void createDBInstance();
    void deleteCloudDBEntity();
    void refreshInfoForDeleteDbInstance();
    void createDBEntityPYSuccess();
    void createDBEntityPYFail();
    void deleteDBEntityPYSuccess();
    void deleteDBEntityPYFail();
    ......
}
```

类DBEntityTask只要方法的含义在表5.3中给出。

表5.3 创建删除实例相关方法

方法	作用
checkCreateStatus()	检查实例的创建状态
selectBestServer()	选择最优的服务器池创建实例
pingServerGroupIP()	Ping服务器组IP，检测是否能ping通

createDbInstance()	创建数据库实例
deleteCloudDBEntity()	删除数据库实例
createDBEntityPYSuccess()	创建实例成功，修改相关数据库表
createDBEntityPYFail()	创建实例失败，修改相关数据库表
deleteDBEntityPYSuccess()	删除实例成功，修改相关数据库表
deleteDBEntityPYFail()	删除实例失败，修改相关数据库表

5.2.2 数据库表设计

云数据库实例的创建比删除的操作要复杂，所涉及的数据库表也要复杂一些，因此这里主要描述实例创建所涉及的数据库表。创建实例时涉及三张表，分别是：表srv_connection、表db_entity_list、表server_pool_list，其中表srv_connection主要记录实例的参数信息如最大连接数、内存、iops，这张主要是用来查询，一般情况不会写入数据，表db_entity_list通过外键max_connection与表srv_connection关联起来，实例的最大连接数确定后会通过表srv_connection来确定其他参数。当确定创建实例的服务器池后，表server_pool_list中的vip字段既是实例的访问ip，记录在表db_entity_list中vip字段里，三种表之间的关系如图5.4所示。

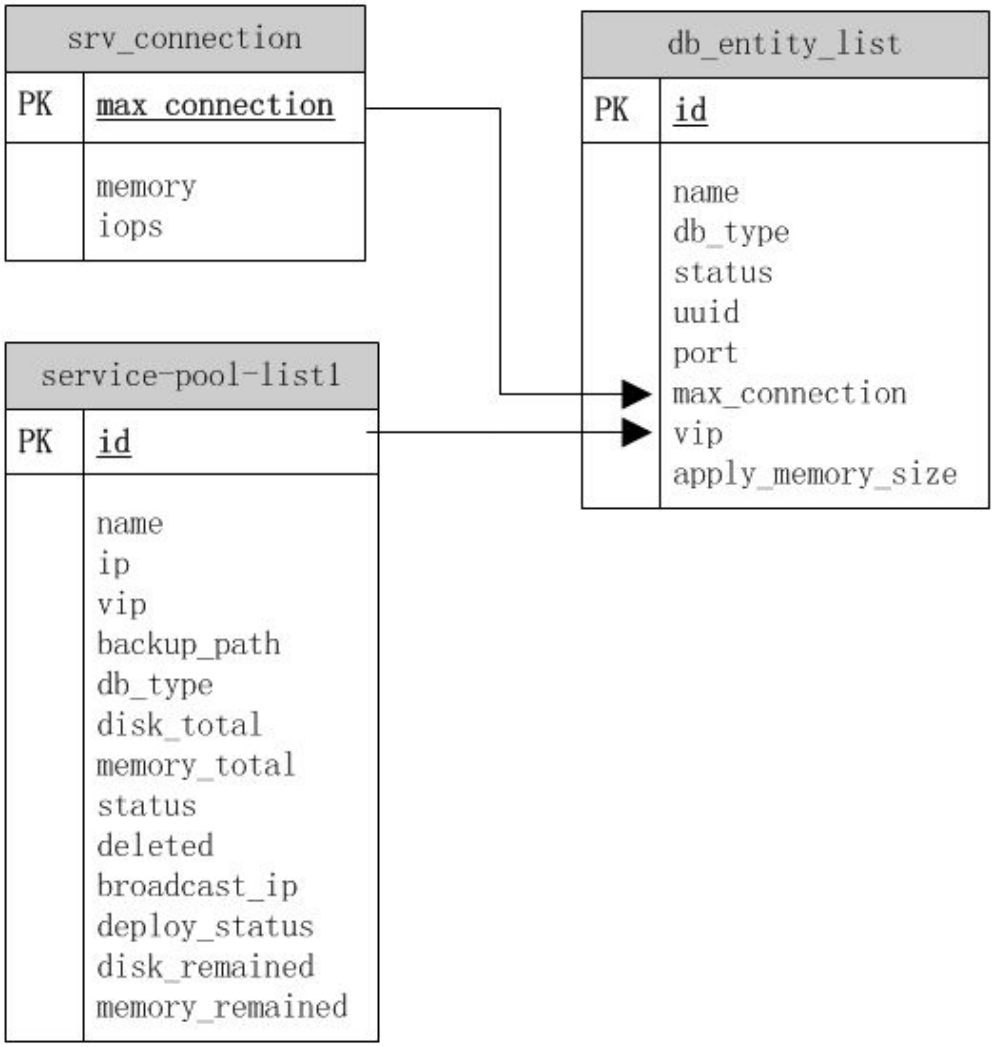

图5.4 创建删除实例数据库表

数据库表service-pool-list已在上一节中给出，数据库表srv_connection和表db_entity_list的详细说明分别见表5.4和表5.5。

表5.4 表srv_connection字段说明

字段名称	数据类型	约束	备注
max_connection	INTEGER	非空，唯一	列表主键
memory	INTEGER		内存大小
iops	INTEGER		iops限制

表5.5 表db_entity_list字段说明

字段名称	数据类型	约束	备注
id	INTEGER	非空，唯一	列表主键
name	VERCHAR(256)	非空	实例名称，长度256
db_type	INTEGER	非空	实例类型
status	INTEGER	无	实例状态（0：不可用，1：可用）
uuid	VERCHAR(256)	非空	为了区别不同类型数据库（实例id无法区分时的情况），长度256
port	INTEGER	无	实例端口号
max_connection	INTEGER	无	实例最大连接数
appy_memory_size	INTEGER	无	实例内存大小，由最大连接数查表得到

5.2.3 具体实现

在创建数据库时，私有云平台会对web页面输入的信息进行前端校验，输入的主要信息参数有最大连接数、实例容量等信息，校验的内容包括：实例名称是否合法、私有网络是否可用等。如果校验失败则提示用户进行更改，如果校验通过，前台会将输入的信息进行编码，以ASN.1的消息格式传给C++端进行处理。

C++端收到ASN.1消息后首先进行解码，并将信息写入表db_entity_list中，将status字段置为不可用，并根据max_connection的值查询数据库srv_connection，获取memory和iops参数。然后会调用刷新物理服务器的方法，获取表server-pool-list中所有此实例类型的服务器池ip地址，并连带实例参数一起以Protocol Buffer的消息格式传入Python端进行处理。Python端通过调用SaltStack的接口，对所有Salt-Minion进行状态信息的获取，并将信息返回至C++端，C++端将获取到的服务器信息更新至server-pool-list表中。这时的server-pool-list表中的物理服务器池的信息都是最新的，通过调用选择最佳服务器池的方法，对服务器池的内存和磁盘剩余大小进行权值计算，选择最优的服务器池进行实例创建，如果所有服务器池均不满足实例创建的要求则返回创建失败。如果满足要求则调用创建实例的方法从数据库表db_entity_list中获取

实例信息，以Protocol Buffer的消息格式传入Python端。

Python端对消息解码后，开始对相应的服务器池进行实例创建，同样是调用SaltStack的接口，Python端会依次完成以下几个操作：

1) 对两个Salt-Minion（即服务器池）所连接的共享存储创建lvm用于安装数据库实例。
2) 在对应LVM（Logical Volume Manager）中创建实例，实例的端口号会根据已存在实例数目依次递增，并生成随机数据库实例访问密码。
3) 将数据库实例配置到PaceMaker中，形成双机热备的集群状态，如图5.5所示。
4) 轮询检测数据库实例状态是否可用，若超过10次检测不可用则创建失败。
5) 重新获取服务器池最新的状态信息。

完成如上操作后，Python端将结果返回至C++端，并完成数据库表db_entity_list的更新，包括vip，用户名，密码等，并将状态置为可用，如果有任何一个步骤出错，都会将状态置为创建失败。具体流程图如图5.6所示。

```
Last updated: Thu Apr 16 05:35:29 2015
Last change: Wed Apr 15 14:01:39 2015 via crm_attribute on node1
Stack: corosync
Current DC: node1 (738527252) - partition with quorum
Version: 1.1.10-42f2063
2 Nodes configured
14 Resources configured


Online: [ node1 node2 ]

p_ip_mysql      (ocf::heartbeat:IPaddr2):       Started node2
p_fs_mysqlbackup        (ocf::heartbeat:Filesystem):    Started node2
 Resource Group: g_mysql3311
     p_fs_mysql3311     (ocf::heartbeat:Filesystem):    Started node2
     p_mysql3311        (ocf::heartbeat:mysql): Started node2
 Master/Slave Set: ms_mysql3311 [p_drbd_mysql3311]
     Masters: [ node2 ]
     Slaves: [ node1 ]
```

图5.5 实例集群配置成功

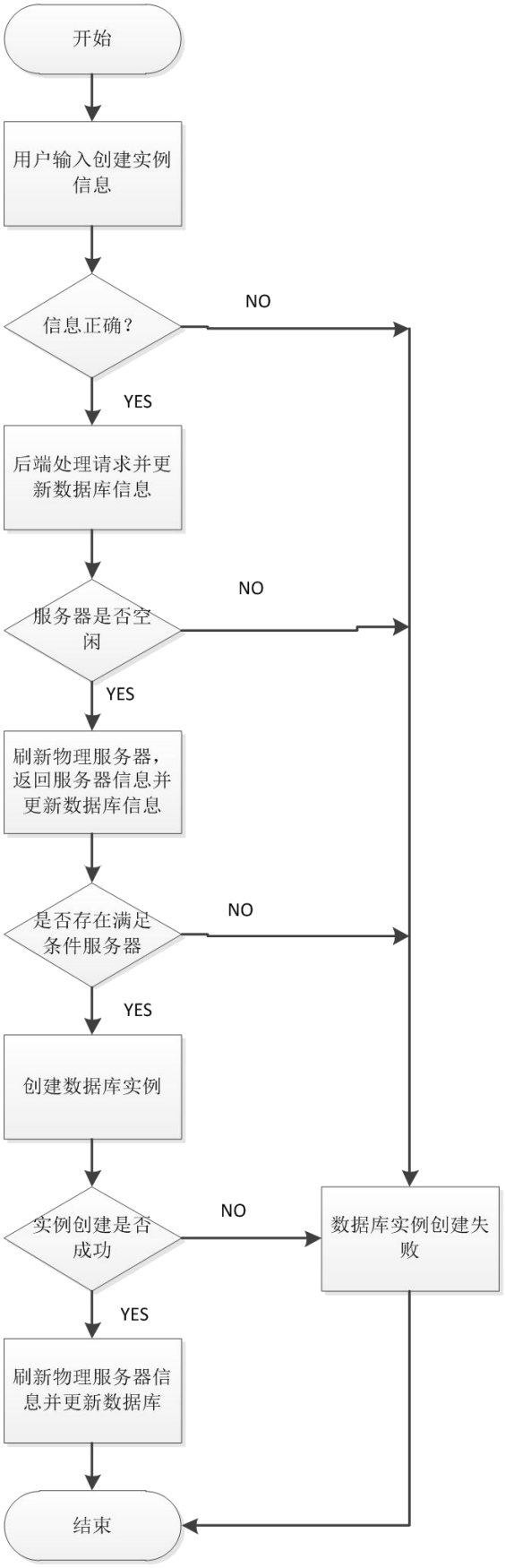

图5.6 创建删除实例流程图

5.3 数据库实例的操作

5.3.1 功能设计

类DBEntityTask中还包含操作实例的相关方法，如修改参数，升级数据库，刷新实例状态等。刷新实例状态主要是用于对实例进项创建或者操作后的状态更新，由于web页面并不会同步更新展示给用户的实例状态，需要用户手动刷新去获取实例最新状态。对数据库实例的操作都是基于已有的并且状态正常的实例，DBEntityTask中关于实例的操作的方法如下所示：

```
class DBEntityTask
{
public:
    ......
    void refreshDBEntityStatus();
    void refreshDBEntityStatusEnd();
    void upgradeDbInstance();
    void controlDBEntity();
    void upgradeDBEntityPYSuccess();
    void upgradeDBEntityPYFail();
    void changeParameter();
    void changeParameterSuccess();
    void changeParameterFail();
}
```

类DBEntityTask中操作实例相关方法的详细说明见表5.6所示。

表5.6 实例操作相关方法

方法	作用
refreshDBEntityStatus()	请求刷新实例状态
refreshDBEntityStatusEnd()	刷新实例状态结束，修改相关数据库表
upgradeDbInstance()	升级数据库实例
upgradeDBEntityPYSuccess()	升级数据库实例成功，修改相关数据库表

upgradeDBEntityPYFail()	升级数据库实例失败，修改相关数据库表
controlDBEntity()	启停数据库实例
changeParameter()	修改实例参数
changeParameterSuccess()	修改实例参数成功，修改相关数据库表
changeParameterFail()	修改实例参数失败，修改相关数据库表

5.3.2 数据库表设计

对实例的操作有升级实例，刷新实例状态和参数更改，其中实例的升级和状态刷新只涉及数据库表db_entity_list中的相关字段，这里不重点讨论。参数修改涉及三张数据库表，分别是：db_entity_list，db-para-list，all-parameter-list。当用户针对某一个实例进行参数修改时，首先在数据库表db_entity_list查询到相关的实例，并根据db_entity_list表中实例的id到db-para-list表中查找db-id字段的值，数据库表db-para-list以字段db-id外键关联至db_entity_list的主键id。在表db-para-list中，每行可修改的参数都会包含uuid属性且值唯一，通过字段uuid外键关联至数据库表all-parameter-list中的主键uuid，这个表里记录了参数值可更改的范围，默认值等信息。修改参数所涉及的三张表之间的关系如图5.7所示。

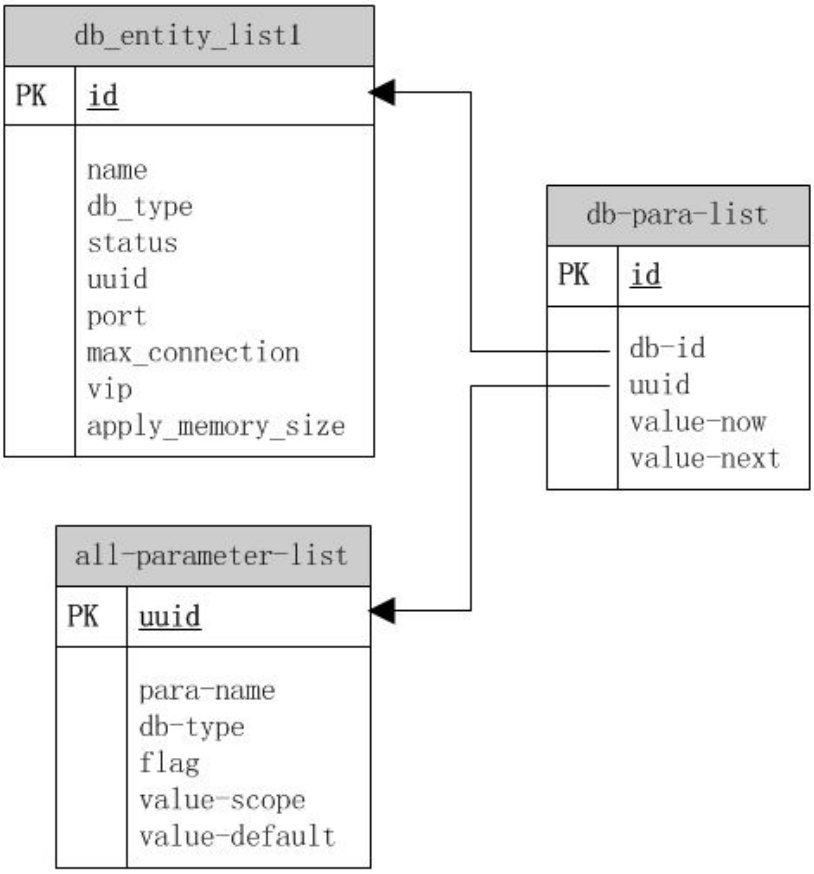

图5.7 修改实例参数相关数据库表

数据库表all-parameter-list和表db-para-list中的字段详细信息分别见表5.7和表5.8所示。

表5.7 表all-parameter-list字段说明

字段名称	数据类型	约束	备注
uuid	VARCHAR(128)	非空，唯一	列表主键，参数uuid
para-name	VARCHAR(256)		参数名称，长度256
db-type	INTEGER	非空	实例类型
flag	INTEGER		修改是否需要重启
value-scope	VARCHAR(128)		参数修改范围
value-default	VARCHAR(128)		参数默认值

表5.8 表db-para-list字段说明

字段名称	数据类型	约束	备注
id	INTEGER	非空，唯一	列表主键
db-id	INTEGER	无	实例id，关联实例信息表的id
uuid	VARCHAR(128)	非空	参数uuid，关联所有参数表中对应实例中可修改参数的uuid

value-now	VARCHAR(128)	无	参数当前值
value-next	VARCHAR(128)	无	用户输入值

5.3.3 具体实现

针对MySQL数据库实例，系统提供了几十种参数的修改，这些参数的预设值以及相关信息都储存在数据库表all-parameter-list中。当用户对某一参数进行修改时，前端会对web页面输入的信息进行校验，如果输入包含非法输入如特殊字符，字符串过长等问题，会在web页面上提示用户重新输入，输入正确后前台会对输入的信息进行处理，以ASN.1的消息编码格式发送至C++端，此时的消息中包含的主要信息有实例id，参数uuid，参数当前值（如果存在当前值），参数修改值等信息。

C++端收到消息后首先进行ASN.1消息解码，根据数据库表all-parameter-list中uuid参数去数据库表db-para-list中进行查询，如果没有查到此uuid即为第一次修改此参数，将此参数的修改信息插入到表db-para-list中，如果非第一修改则直接更新数据库表db-para-list中修改值（value-next）。由于数据库实例的参数有些可以在实例运行的状态下动态修改，有些则需要把实例停止之后再进行修改，所以系统需要通过查询表all-parameter-list中所修改参数对应的flag参数的属性来确定是否需要重启，获取到这些参数后，会将消息进行Protocol Buffer消息格式进行编码，发送至Python端，此时消息中包含的主要信息有实例名，参数修改值，是否需要重启，服务器池ip等信息。

Python端收到消息后首先进行Protocol Buffer消息解码，然后根据消息中的flag的值来判断修改实例参数是否需要重启。如果不需要重启，通过进入MySQl实例执行响应的修改参数的sql语句就可以成功修改实例参数，如果需要重启，首先要关闭实例，待检测实例完全关闭后再对实例的配置文件进行修改，修改完成后重新启动实例，轮询十次检测实例状态，当检测到实例启动完成之后，将消息返回至C++端，C++修改数据库表db-para-list中的当前值（value-now），并将数据库实例状态置为可用。具体流程图如图5.8所示。

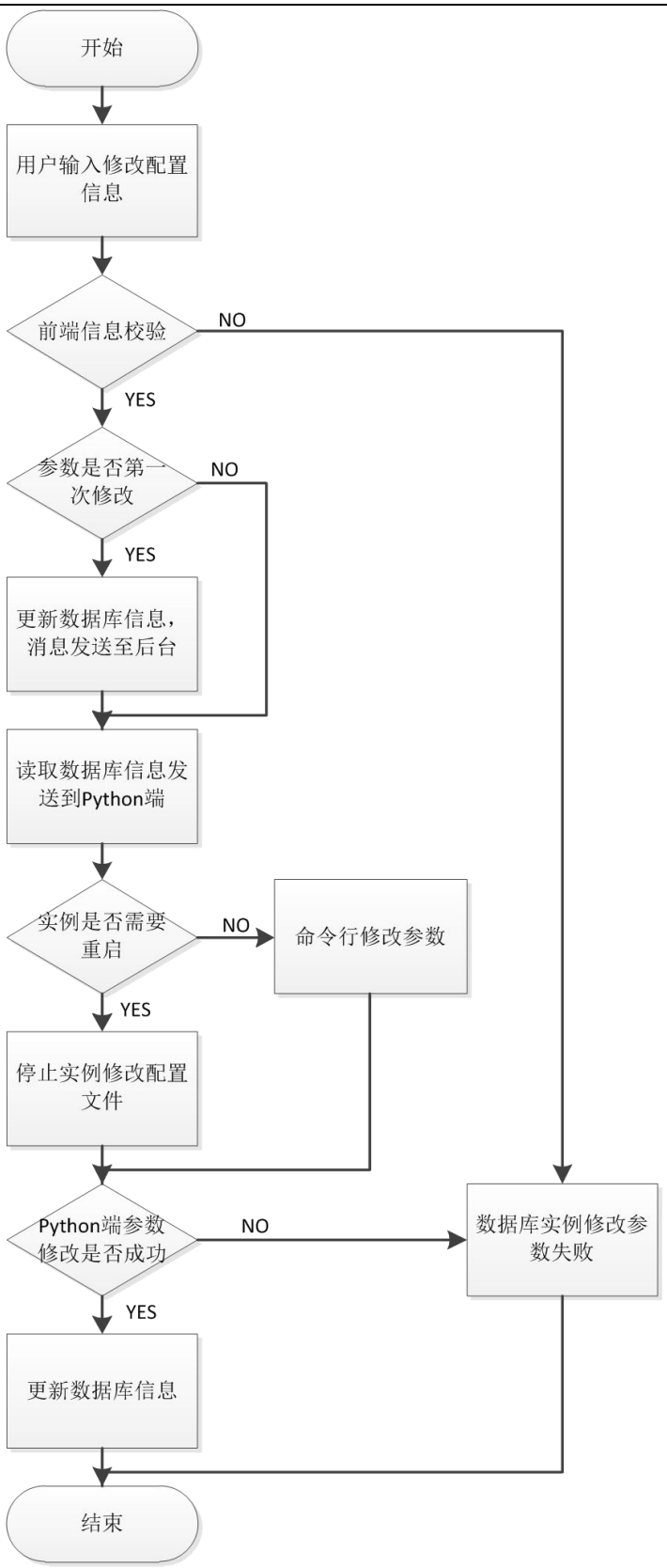

图5.8 修改实例参数流程图

5.4 数据库实例备份与恢复

5.4.1 功能设计

数据库实例的备份和恢复的相关方法在类DBEntityTask中定义，实例的备份分为两种：立即备份和定时备份，定时备份是备份策略中采用的方式，用户可以自己定义周一到周日的备份策略，灵活高效的进行数据库实例备份。DBEntityTask类中包含的关于备份与恢复的方法主要有立即备份实例，删除备份，恢复实例，创建删除临时实例等。临时实例是基于备份集或七天内的一个时间点的备份恢复出来的一个临时的云数据库实例，是实例备份恢复的一种方式，这里不再单独介绍。类DBEntityTask中关于备份与恢复的主要方法如下所示：

```
class DBEntityTask
{
public:
    ......
    void backupCloudDBEntityNow();
    void recoverDBEntity();
    void backupDBEntityPYSuccess();
    void backupDBEntityPYFail();
    void recoveryDBEntityPYSuccess();
    void recoveryDBEntityPYFail();
    void deleteBackupFile();
    void deleteBackupFileSuccess();
    void createTempDbInstance();
    void createTempDbInstanceSuccess();
    void createTempDbInstanceFail();
    ......
}
```

类DBEntityTask中关于备份与恢复的主要方法的详细说明见表5.8所示。

表5.8 实例备份恢复相关方法

方法	作用
backupCloudDBEntityNow()	立即备份数据库实例
recoverDBEntity()	恢复数据库实例
backupDBEntityPYSuccess()	备份数据库实例成功，修改相关数据库表
backupDBEntityPYFail()	备份数据库实例失败，修改相关数据库表
recoveryDBEntityPYSuccess()	恢复实例备份成功，修改相关数据库表
recoveryDBEntityPYFail()	恢复实例备份失败，修改相关数据库表
deleteBackupFile()	删除备份文件
deleteBackupFileSuccess()	删除备份文件成功，修改相关数据库表
createTempDbInstanceFail()	使用备份文件创建临时实例

5.4.2 数据库表设计

数据库实例的备份与恢复涉及到两张数据库表分别是backup_info_list和backup_stra_list，数据库表backup_stra_list主要用于备份策略的设置，用户可以自定义周几进行备份，备份的方式是增量备份还是全量备份，这些信息都会记录在表backup_stra_list中，当到达用户设置的时间时，实例就会自动备份。数据库表backup_info_list主要用于记录实例备份文件的详细信息，无论是备份方式是手动备份还是自动备份，都会将备份的文件信息记录在表backup_info_list中，用type自动去区别备份方式，用backup_type区别备份的类型（增量备份或者全量备份）。两张表之间以实例的id（字段db_id）关联到对应的实例。涉及的数据表如图5.9。

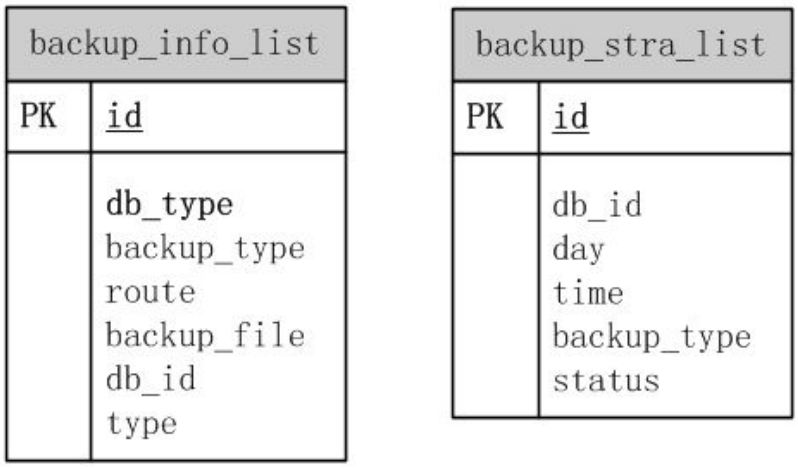

图5.9 备份恢复相关数据库表

数据库表backup_info_list和backup_stra_list的字段信息见表5.9和表5.10。

表5.9 表backup_info_list字段说明

字段名称	类型	约束	备注
id	INTEGER	非空，唯一	实例备份策略表主键
db_id	INTEGER	无	设置备份策略的实例id
day	INTEGER	无	自动备份的日期，以一周为周期
time	VARCHAR(128)	无	自动备份的时间，24小时计时
backup_type	INTEGER	无	备份类型，有增量和全量两种
status	INTEGER	无	此备份策略是否可用，1为可用，0为不可用

表5.10 表backup_stra_list字段说明

字段名称	数据类型	约束	备注
id	INTEGER	非空，唯一	备份详情表主键序号
db_type	INTEGER	非空	实例类型
backup_type	INTEGER	无	备份类型，全量或增量
route	VARCHAR(256)	无	备份路径
backup_file	VARCHAR(256)	无	备份文件名称
db_id	INTEGER	无	备份实例id
type	INTEGER	无	自动备份或者手动备份

5.4.3 具体实现

（1）实例的备份

数据库实例的备份有两种触发方式：用户手动备份和定时器触发的自动备份。定时器触发的自动备份实在备份策略中进行设置的，当用户设置好备份策略后，详细信息会存储在数据库表backup_stra_list中，当到达表中记录的备份时间时，定时器就会触发备份。当触发备份请求时，前端首先回去检测实例状态和实例是否正在备份，如果不可用或者正在备份则无法备份，如果状态可用则把消息以ASN.1的编码格式发送至后台C++端。此时的消息中包含的主要信息有：备份类型（全量或者增量备份），备份路径，备份方式（自动备份或者手动备份）。

C++后台收到消息后首先进行ASN.1消息解码，然后连接数据库，将备份信息写入表backup_info_list中。由于增量备份必须是在全量备份的基础上进行的，所有要根据备份类型

（字段type）判断是增量备份还是全量备份，如果是全量备份则直接进行备份操作，如果是增量备份系统会查到此实例在表backup_info_list中是否存在历史全量备份，存在全量备份会在最近的一个全量备份的基础上进行增量备份，不存在历史全量备份则将本次增量备份强制转变为全量备份。C++端将备份所需要的信息进行Protocol Buffer消息格式编码，发送到Python端（Salt-Master）进行处理。此时消息中包含的主要信息有：备份类型（全量或者增量备份），备份路径，备份方式（自动备份或者手动备份）、实例名称等信息。

Python端收到消息后进行Protocol Buffer解码，根据备份类型进行备份，备份的位置是在共享存储中为备份所划分的VG中，备份文件的名称（backup_file）为实例名称加时间戳，备份工具使用的是innobackupex，当检测到系统备份完成后，Python端将备份信息重新返回至Python端，这时的信息中会包含备份路径（route）已经备份文件的名称（backup_file），并写入数据库表backup_info_list中，这时用户可在web页面查询到实例备份文件的记录。实例备份流程图如图5.10所示。

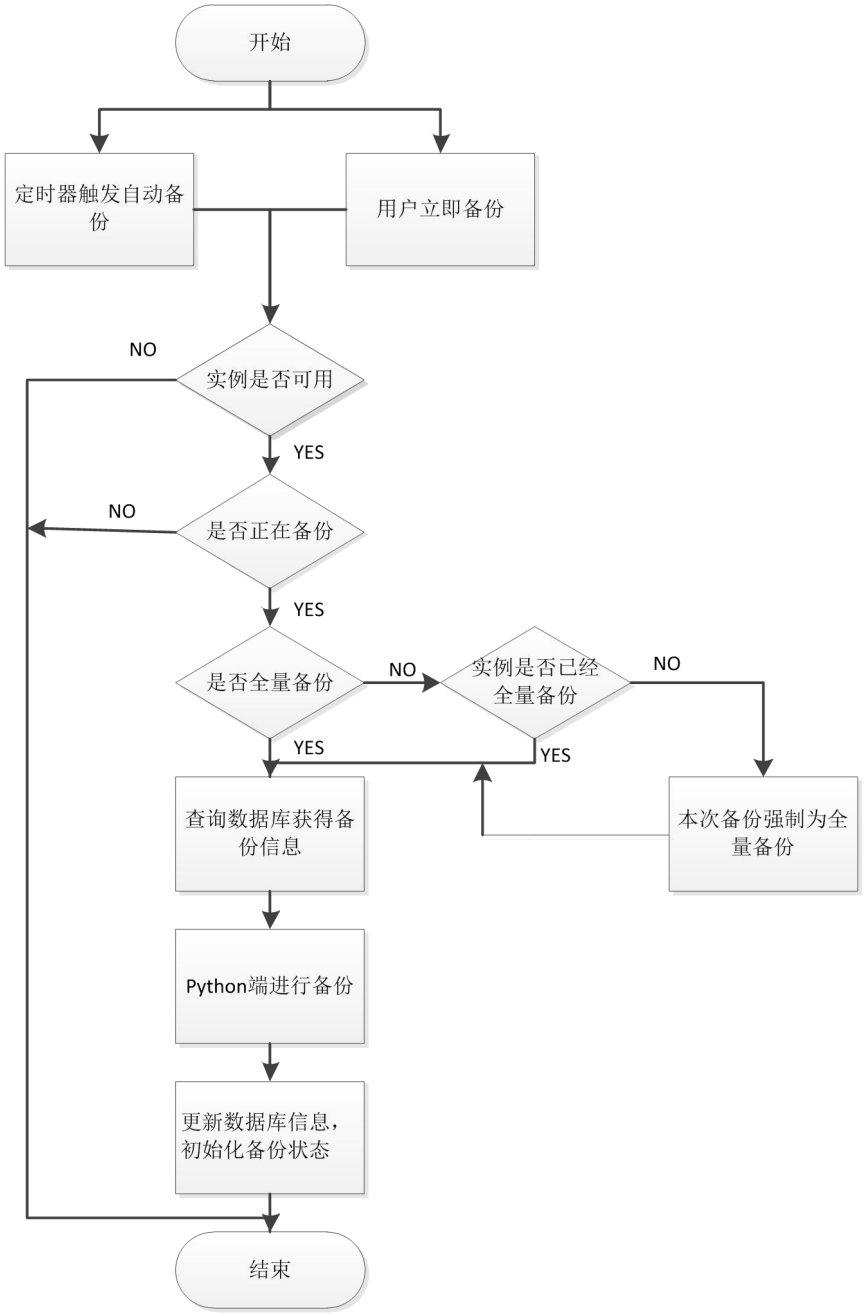

图5.10 实例备份流程图

（2）实例的恢复

在用户制定备份策略后，数据库实例会按照备份策略的方式进行自动备份，由于自动备份的备份集在系统中只能保存十五天，所以在恢复实例增量备份时需要按照一定的规则检测之后再恢复。当自动备份的备份集到达十五天，系统会先检测backup_stra_list中的备份信息，通过备份时间（type）来判断是否过期，由于增量备份要依赖最近的全量备份，所以在检测备份文件是否需要删除时，需要考虑当前全量备份的备份时间之后是否存在没有到期的增量备份，如果存在则保留此全量备份，不存在则从数据库表backup_stra_list和backup_info_list删除此备份文件。

当恢复全量备份时，只需要查询到此全量备份的备份文件，然后直接恢复即可。当恢复增量备份时，系统会把数据库backup_info_list中备份信息读取出来，并且倒序排列，找出距离需要恢复的增量备份最近的一个全量备份，先恢复此全量备份，之后依次恢复增量备份，直到把当前增量备份恢复完成。

5.5 数据库性能监控告警

5.5.1 功能设计

性能监控告警分为两部分，一部分是对实例的监控，包括监控实例的性能参数，命中率等，这部分的相关方法定义在类DBEntityTask中，另一部分是对服务器资源信息的监控，包括磁盘使用率，内存使用率，CPU使用率等，这部分的相关方法定义在类PhysicalServerTask中。这两个类的具体设计如下：

```
class DBEntityTask
{
public:
    ......
    void monitDBEntityStatus();
    void monitDBEntityStatusEnd();
    ......
};
class PhysicalServerTask
{
public:
    ......
    void timerCallBack();
    Static int refreshPhysicalServerInfoReq();
    Static int dealRefreshPhysicalServerInfoResp();
    void setAlarmThreshold();
    ......
}
```

类DBEntityTask和类PhysicalServerTask中关于性能监控的方法说明如表5.11所示。

表5.11 实例监控告警相关方法

方法	作用
monitDBEntityStatus()	监控实例状态
monitDBEntityStatusEnd()	监控实例状态结束，修改相关数据库表
timerCallBack()	定时器触发
refreshPhysicalServerInfoReq()	刷新物理服务器信息请求
dealRefreshPhysicalServerInfoResp()	处理刷新返回结果
setAlarmThreshold()	设置阈值告警

5.5.2 数据库表设计

性能的监控告警涉及三张数据库表，分别是表db_entity_list，表db-alarm，和表all-alarm。表db-alarm存储这所有需要监控的参数信息，通过字段db_uuid与表db_entity_list中实例关联起来。数据库表all-alarm中存储的是每个参数的监控值信息和告警范围，它的主键uuid外键关联至db-alarm中的alarm_uuid，字段db_type与表db_entity_list相关联。系统每隔一段时间就进行一次信息采集，通过一定的逻辑处理方式存储到数据库表db-alarm中，然后根据数据库表all-alarm的值确定是否需要告警。三张表之间的关系见图5.11所示。

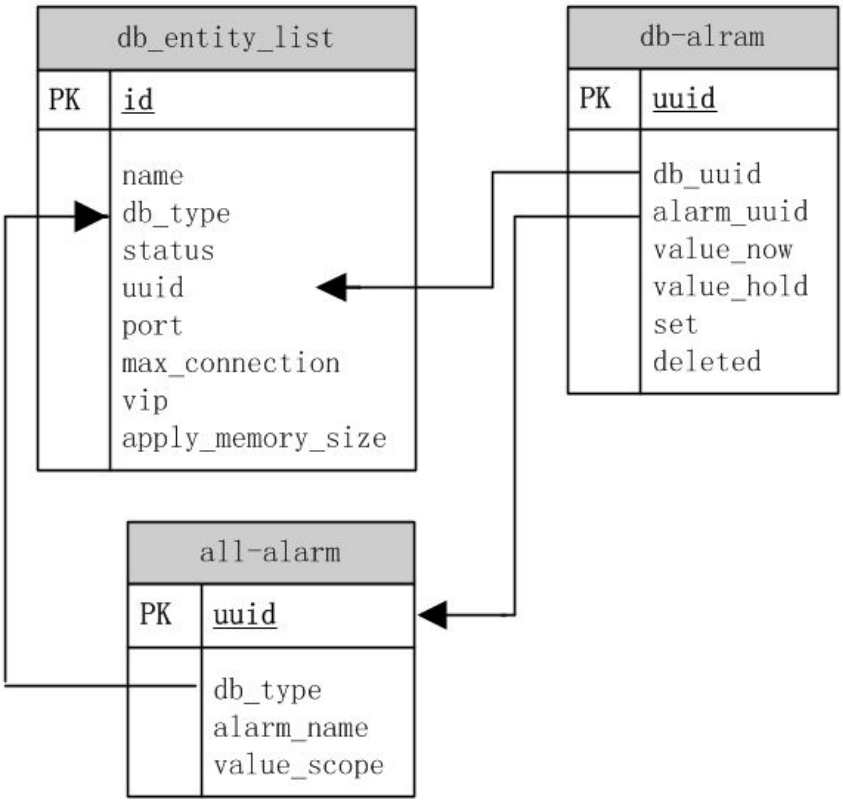

图5.11 监控告警相关数据库表

数据库表db-alarm与表all-alarm中的字段属性说明分别见表5.12和5.13所示。

表5.12 表db-alarm字段说明

字段名称	数据类型	约束	备注
uuid	VARCHAR(128)	非空，唯一	列表主键，参数uuid
alarm_uuid	VARCHAR(128)	非空	监控项唯一标识
value_now	VARCHAR(128)	无	参数当前值
value_hold	VARCHAR(128)	无	参数维持值
deleted	INTEGER	无	是否已删除
db_uuid	VARCHAR(128)	非空	关联至实例uuid

表5.13 表all-alarm字段说明

字段名称	数据类型	约束	备注
uuid	VARCHAR(128)	非空，唯一	列表主键，参数uuid
db_type	INTEGER	非空	关联至实例类型
alarm_name	VARCHAR(256)	无	监控项名称
value_scope	VARCHAR(256)	无	监控项可波动范围

5.5.3 具体实现

数据库实例监控告警模块可以分为两个子模块，即监控和阈值告警。用户首先要在web页面设置需要监控的参数，以及需要告警的阈值，定时器每隔5分钟对数据库实例监控采集一次，监控的结果以键值对的形式插入到mongodb数据库中。监控实例时需要运行sql查询相应的参数值，监控服务器时运行Linux命令进行查询，如top，iostat，df等命令。将存储在mongodb里的数据以一定的数据结构进行储存，然后以Protocol Buffer的消息编码格式发送至C++端。

C++端收到消息后首先进行Protocol Buffer解码，然后查询数据库表all-alarm的信息判断消息中是否存在监控项的至大于表all-alarm中的阈值范围（value_scope），如果大于此阈值，将告警信息写入表db-alarm中，用第三方告警模块进行告警，用户可以在web页面上查询到相关的告警信息。具体流程图见图5.12所示。

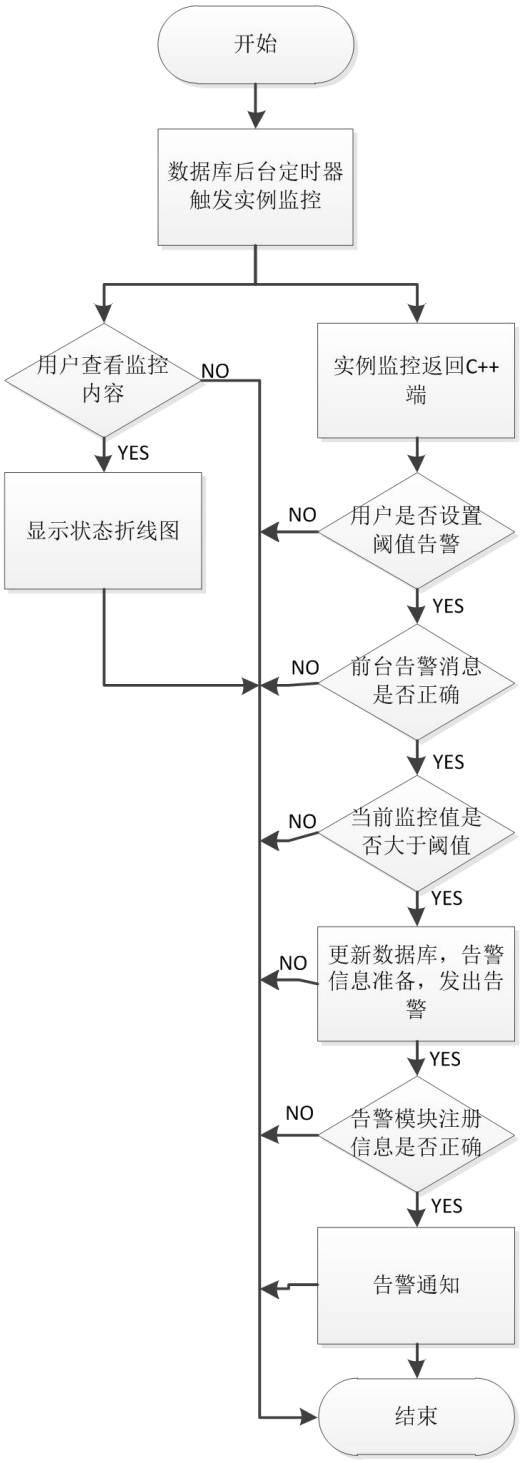

图5.12 实例监控告警流程图

5.6 本章小结

本章主要介绍了云数据库具体的设计实现，每节内容分为三块：功能设计，数据库表设计，和关键技术的实现流程。这一章节与第三章节的需求分析相对应，对主要的需求的实现进行了详细的说明，包括云数据库服务器池的相关操作，数据库实例的创建和删除，实例参

数的修改，数据库实例的备份与恢复，还有数据库的监控和告警。本章以数据库关系图，流程图，表格的形式去展现每个功能的实现，以图文的形式尽可能展现设计的关键技术点。

第六章 系统的测试验证

6.1 测试环境的准备

本次测试环境搭建是基于共享存储的云数据库环境搭建，一共需要4台服务器和一台共享存储服务器，用来安装云平台，SaltStack，云数据库服务器池，具体的服务器信息以及IP分配见表6.1所示：

表6.1 测试环境

服务器(名称)	作用	系统	IP分配
服务器1(CSM)	安装云平台	windows/Linux	172.5.8.15
服务器2(Master)	作为Salt-Master	CentOS 6.5	172.5.8.19
服务器3(Mysql1)	数据库服务器池(Salt-Minion)	CentOS 6.5	172.5.8.77
服务器4(Mysql2)	数据库服务器池(Salt-Minion)	CentOS 6.5	172.5.8.78
共享存储	储存数据库文件	预装存储系统	172.70.0.200

测试环境搭建完成之后，即可通过云平台的web页面进行云数据库的相关操作，本次测试验证云数据库的两个主要功能：添加物理服务器池和创建数据库实例，并对系统的高可用性进行验证。

6.2 功能测试

6.2.1 测试添加物理服务器池

进入云平台web页面数据库页面，点击添加物理服务器池会弹出页面提示服务器池，按照要求填写，如图6.1所示，其中服务器1和服务器2即为环境搭建中准备的两个数据库物理服务器3和4，按照环境规划中的IP地址，以及服务器名称填写到相应的表格中。虚IP地址为与服务器池同一网段，但未使用过的IP地址即可。组播地址填写合法的地址范围，并且未曾使用过即可。备份路径按照个人需求填写。信息填写完成后点击确定，系统便会发送请求自动添加物理服务器池。

图6.1 添加服务器池web弹窗

数据库服务器池添加成功后会页面提示服务器池可用，并显示出相关信息，如图6.2所示。

图6.2 添加服务器池成功web页面

进入数据库物理服务器，可以查看到如图6.3所示集群状态。

```
Last updated: Fri Dec 19 23:46:02 2014
Last change: Fri Dec 19 23:45:50 2014
Stack: classic openais (with plugin)
Current DC: Mysql1 - partition with quorum
Version: 1.1.11-97629de
2 Nodes configured, 2 expected votes
6 Resources configured


Online: [ Mysql1 Mysql2 ]

p_ip_mysql      (ocf::heartbeat:IPaddr2):       Started Mysql1
p_fs_mysqlbackup        (ocf::heartbeat:Filesystem):    Started Mysql1
```

图6.3 服务器池集群状态

综上，根据测试可以得出，添加物理服务器功能正常。

6.2.2 测试创建数据库实例

物理服务器添加完成后就可以创建数据库实例，同样进入云平台web页面，点击新建实例会弹出新建实例的弹框，如图6.4所示。填完信息之后点击提交系统会自动发送请求创建实例。

图6.4 创建数据库实例web弹窗

数据库实例添加成功后在web页面上状态显示可用，如图6.5所示。

图6.5 创建实例成功web页面

在数据库物理服务器上可以查看到数据库实例Mysql3312的集群状态为正常，并且启动在服务器Myql1上，如图6.6所示。

```
Last updated: Sat Dec 20 23:20:35 2014
Last change: Sat Dec 20 03:17:16 2014
Stack: classic openais (with plugin)
Current DC: Mysql1 - partition with quorum
Version: 1.1.11-97629de
2 Nodes configured, 2 expected votes
6 Resources configured


Online: [ Mysql1 Mysql2 ]

p_ip_mysql       (ocf::heartbeat:IPaddr2):       Started Mysql1
p_fs_mysqlbackup        (ocf::heartbeat:Filesystem):    Started Mysql1
 Resource Group: g_mysql3311
     p_fs_mysql3311      (ocf::heartbeat:Filesystem):    Started Mysql1
     p_mysql3311         (ocf::heartbeat:mysql): Started Mysql1
 Resource Group: g_mysql3312
     p_fs_mysql3312      (ocf::heartbeat:Filesystem):    Started Mysql1
     p_mysql3312         (ocf::heartbeat:mysql): Started Mysql1
```

图6.6 服务器集群实例状态

下面我们在服务器Mysql1上验证实例是否可以正常使用，输入“mysql -uroot -P3312 -h172.5.8.248 -p+ password”，根据显示进入实例成功，说明创建的实例是可用的，如图6.7所示。

```
[root@Mysql1 ~]# mysql -uroot -P3312 -h172.5.8.248 -p7WezN4FPHcSXqOKf
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 5.6.19-0ubuntu0.14.04.1-log (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.01 sec)

mysql>
```

图6.7 验证实例是否可用

下面通过强行使处于active状态数据库服务器宕机，即Mysql1，测试双机热备高可用方案书否生效，实例能否切换至备机Mysql2上，切换结果如图6.8所示。

```
Last updated: Sat Dec 20 23:26:22 2014
Last change: Sat Dec 20 23:26:12 2014
Stack: classic openais (with plugin)
Current DC: Mysql1 - partition with quorum
Version: 1.1.11-97629de
2 Nodes configured, 2 expected votes
6 Resources configured


Node Mysql1: standby
Online: [ Mysql2 ]

p_ip_mysql      (ocf::heartbeat:IPaddr2):       Started Mysql2
p_fs_mysqlbackup        (ocf::heartbeat:Filesystem):    Started Mysql2
 Resource Group: g_mysql3311
     p_fs_mysql3311     (ocf::heartbeat:Filesystem):    Started Mysql2
     p_mysql3311        (ocf::heartbeat:mysql): Started Mysql2
 Resource Group: g_mysql3312
     p_fs_mysql3312     (ocf::heartbeat:Filesystem):    Started Mysql2
     p_mysql3312        (ocf::heartbeat:mysql): Started Mysql2
```

图6.8 测试实例高可用性

根据切换结果可以看出，实例切换到了备机Mysql2上，说明高可用方案已经生效。根据以上测试结果可以得出，本系统可以正常创建具有高可用性的MySQL数据库实例，功能正常。

6.3 本章小节

本章节对系统进行了功能性测试验证，首先对测试环境的部署进行了描述，然后测试了云数据库的两个主要功能：添加服务器池和新建云数据库实例。根据测试结果可能得出两个功能均正常，并且数据库实例的双机热备高可用性也能够正常生效。

第七章 总结与展望

云数据库在云计算技术迅猛发展且日趋成熟的情况下诞生的，与其他基于IaaS的云计算资源一样，云数据库技术的出现给企事业单位的发展带来了切实的好处。企业无需再为购买大量昂贵的服务器，以及部署拥有大量服务器的机房环境，通过云端的网络，主机，数据库去构建自己的数据中心。本章对云数据库项目开发过程中遇到的问题进行总结，并提出项目的不足，以及后期需要继续完善的地方。

7.1 云数据库项目的总结

7.1.1 论文中遇到的问题及解决方法

项目在开发过程中遇到了不少困难，根据市场对云数据库的需求，云数据库项目需要解决的问题主要有以下三点：

（1）高可用性：无论对于企业还是产品，对数据库的使用都是极其频繁的，因此必须要保证数据库服务器最大限度的可用性。

（2）自动化部署：为了节约人力成本，简化数据库安装操作的复杂度，可以大大降低DBA的工作量，因此要实现数据库的一键式安装，部署，创建等一系类操作。

（3）数据隔离：由于存在多个部门使用同一个数据库服务器的情况，因此要做到不同部门的实例相互隔离。

以上三个问题的解决贯穿了整个项目的研发，下面针对每个问题的解决方式进行总结性的概括：

（1）高可用性的实现：本项目中高可用性主要是采用的双机热备的集群设计方式，通过Pacemaker、corosync和共享存储相结合的方式实现的。在本文的第二章和第三章中对集群的高可用性进行了介绍，并重点介绍了Pacemaker和共享存储方式的实现。

（2）自动化部署的实现：本项目中通过对开源工具SaltStack的使用来实现自动化部署的需求，SaltStack是集群运维工具，在本文的第四章中对SaltStack的架构及工作方式进行了详细的介绍。

（3）数据隔离：数据的隔离又称租户的隔离，本文从两个方面去实现的，首先是在系统层面进行隔离，对Linux系统采用多用户的形式进行实例创建，每个实例对应一个Linux用户。

然后是数据存储层面的隔离，在每个实例创建时，都在共享存储中为每个实例划分出一个逻辑卷LVM，这样就实现的租户的完全隔离。在本文第五章实现实例创建时对实现过程进行了具体的描述。

7.1.2 云数据库项目的不足

云数据库项目虽然实现了主要的功能需求，但由于开发周期短，仍存在一些不足需要后续继续进行完善，主要遗留问题如下：

（1）数据安全问题：本项目中云数据库的物理服务器池采用了双机热备的实现方式，这种实现方式可以保证物理服务器的高可用性，但是在数据存储方面本项目中采用的是共享存储的方案，主备服务器共享同一块存储，在这种设计下，如果存储介质出现了问题则会造成数据的永久性丢失，因此后续开发过程中应考虑介质容灾的实现。

（2）网络传输问题：与本地机房对比，使用云端数据库要依赖于网络环境，很容易出现延迟等问题，当大并发数据量访问数据库时往往会造成性能问题。如果创建较大容量的实例，且网络环境不是很好的情况下，会出现等待时间过长的情况，这样大大影响了用户的体验，因此后续开发需要对网络较差的情况下的传输进行优化，保证用户的良好体验性。

（3）兼容性问题：数据库的类型很多，主要分为关系型数据库和非关系型数据库[40]。本系统主要支持关系型数据库，如MySQL，Oracle，SQL Server。但是对非关系型数据库的需求量依然很大，如MongoDB，Redis，Memcached等，后续开发需要依次支持。

7.2 云数据库技术的展望

云数据库是云计算资源池化技术的一部分，因此云数据库的发展依赖于云计算的发展，根据最近几年云计算的发展状况，可以预料以后云数据库的运用会越来越广泛。中小企业出于成本考虑，会选择公有云数据库，而大公司或者事业单位会选择为自己公司构建私有云数据库。

云数据库的安全问题会成为近几年解决的首要问题，并逐渐消除云数据库的安全隐患。随着市场的需求不断扩大，越来越多类型的数据库会移动到云端，所以会诞生出大量的云数据库解决方案。使用云数据库带来的好处是显而易见的，相信未来云数据会有更广阔的发展前景。

7.3 本章小结

本章讨论了云数据库给企事业单位的带来的优势，从公有云和私有云两个方面进行了分析。并对当前云数据库技术存在的问题进行了简要的描述。从现在的国内市场看公有云的代表有阿里云，百度云盘等，私有云有华为企业云，H3C CSM等，这些代表均与人们的工作生活密切联系在一起，本章最后以这些企业为基准，结合目前的技术对未来云数据库的发展进行了展望，随着云计算的普及，云数据库技术也会发展的越来越好。

参考文献

[1] 朴雅宁. 云计算在教育领域的应用[J]. 信息与电脑 (理论版), 2013, 3: 128.

[2] 任蔷. 基于云计算的教育资源评价方法研究[D]. 西安电子科技大学硕士论文, 2014.

[3] 刘彬. 面向云环境可信任根相关关键技术研究[D]. 北京邮电大学, 2012.

[4] 亚马逊推出管理方面数据库服务[J]. 硅谷, 2012.

[5] 郑昌兴. 云计算军事应用研究[J]. 国防科技, 2011, 32(6): 22-26.

[6] 王洋洋. 针对 NFC 技术的云端体系架构研究[D]. 西南大学, 2013.

[7] 王建峰, 樊宁, 沈军. 电信行业云计算安全发展现状[J]. 信息安全与通信保密, 2012 (11): 98-101.

[8] Group A.Amazon elastic compute cloud (Amazon EC2). http: // aws.amazon.com/ec2 . 2010

[9] 赵少卡, 李立耀, 凌晓, 徐聪, 杨家海. 基于 OpenStack的清华云平台构建与调度方案设计[J]. 计算机应用, 2013, 33(12): 3335-3338.

[10] 周生佩. 云数据库服务管理研究与实现[D]. 华中科技大学, 2013

[11] Pepple K. Deploying openstack[M]. " O'Reilly Media, Inc.", 2011.

[12] 张园, 张云勇, 房秉毅. 一种基于工作流的云系统自动化部署架构[J]. 电信科学, 2014, 30(11): 14-21.

[13] 李小宁, 李磊, 金连文, 等. 基于 Open Stack 构建私有云计算平台[J]. 电信科学, 2012, 28(9): 1-8.

[14] 代码改变世界. http://blog.csdn.net/lynn_kong/article/list/4, 2012.10.18

[15] Rosado T, Bernardino J. An overview of openstack architecture[C]//Proceedings of the 18th International Database Engineering & Applications Symposium. ACM, 2014: 366-367.

[16] 王朋涛. 大规模 OpenStack 集群自动化部署与系统管理研究[D]. 杭州电子科技大学, 2014.

[17] Khan R H, Ylitalo J, Ahmed A S. OpenID authentication as a service in OpenStack[C]//Information Assurance and Security (IAS), 2011 7th International Conference on. IEEE, 2011: 372-377.

[18] Szyrkowiec T, Autenrieth A, Gunning P, et al. First field demonstration of cloud datacenter workflow automation employing dynamic optical transport network resources under OpenStack & OpenFlow orchestration[C]//39th European Conf. and Exhibition on Optical Communication (ECOC). 2013: 22-26.

[19] SaltStack. http://www.Saltstack.com/.

[20] Hosmer B. Getting started with Salt stack--the other configuration management system built with Python[J]. Linux journal, 2012, 2012(223): 3.

[21] Rostanski M, Grochla K, Seman A. Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ[C]//Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on. IEEE, 2014: 879-884.

[22] 关于RabbitMQ, http://lynnkong.iteye.com/blog/1699684, 2014-2-12.

[23] Jones B, Luxenberg S, McGrath D, et al. RabbitMQ Performance and Scalability Analysis[J]. project on CS, 4284.

[24] Hintjens P. ZeroMQ: Messaging for Many Applications[M]. " O'Reilly Media, Inc.", 2013.

[25] 杨晓芬, 王永会, 刘轶. 实时数据库系统双机热备机制设计与实现[J]. 计算机工程与应用, 2012, 48(29): 124-127.

[26] 关于集群技术的几个新工具的介绍, http://blog.csdn.net/superch0054/article/details/9664313, 2013.

[27] MySQL技术内幕-InnoDB存储引擎, http://www.docin.com/p-368854327.html, 2013.01.02.

[28] 剖析数据库性能调优技术之索引调优, http://blog.csdn.net/lcj8/article/details/2069418.

[29] 魏少峰, 张颖. 对计算机数据库备份与恢复技术的研究[J]. 科技风, 2012 (6): 71-71.

[30] 张晨. 浅析计算机数据库备份与恢复技术的应用[J]. 中国新通信, 2015, 17(10): 76-76.

[31] DB2体系架构, http://blog.sina.com.cn/s/blog_5fdcb4a00100er2r.html.

[32] 深入浅出：OpenStack开源云计算架构详解_品高云计算官方博客,

http://blog.sina.com.cn/s/blog_669fa76a01011iwg.html.
[33] OpenStack. http://www. openstack. org .
[34] 黄梁, 陈鲁敏, 王加兴, 等. 企业私有云平台建设研究[J]. 机电工程, 2014, 31(8): 1090-1093.
[35] 金誉华. 校园教学云计算平台的构筑设计[J]. 陕西交通职业技术学院学报, 2015 (3): 22-26.
[36] 张微微. 虚拟桌面技术在高校公共机房管理中的应用[J]. 软件导刊, 2015, 14(11): 197-199.
[37] Myers C. Learning Saltstack[M]. Packt Publishing Ltd, 2015.
[38] 李春青, 李海生. Web 数据库技术及其发展趋势[J]. 软件导刊, 2012, 11(2): 155-156.
[39] 刘雪飞, 吴伯桥, 凌涛. ASN. 1 在网络管理中的应用研究[J]. 信息安全与技术, 2013, 4(6): 95-97.
[40] Ferretti L, Colajanni M, Marchetti M. Supporting security and consistency for cloud database[M]//Cyberspace Safety and Security. Springer Berlin Heidelberg, 2012: 179-193.

致谢

三年的研究生生涯转瞬即逝，这段时间让我成长为一个可以立足社会，勇于逆流而上的人。无论是在学习工作上，还是在科学研究上都有了很大的进步，也让我对自己有了更深层次的认知和定位。在本论文的写作过程中也遇到了很多困难，最后在大家的帮助下一一解决，所以我需要感谢的人很多，感谢你们对我的谆谆教诲和耐心指导。

首先要感谢的是我的导师钱学荣教授，从您给我们讲课的细致认真和对待学术的态度让我学会了严谨的做人做事。在研究生学习期间，钱老师鼓励学生全面发展，在兼顾科研的同时支持学生进入企业实践，让我们的理论知识得到更好的运用。在本论文开题时，钱老师也针对开题给出了细致的修改意见。后续论文写作过程中钱老师也积极的指导督促我进行科学研究以及论文修改。无论是学术还是生活，钱老师都给了我极大的帮组，在这里我衷心的对您说一声：谢谢您！

然后我要感谢的是我企业导师禹龙，是您细致耐心的教导让我从初入公司的一无所知成长能够参与完成本论文项目的开发。还要感谢项目组的同事们，是和你们一起愉快的合作才让项目顺利的上线。还要谢谢师门和宿舍的小伙伴们，是你们陪我一路走过这三年，带给我各种开心和快乐。

还要感谢我的父母，是您们一如既往的支持，才能让我不断前行，敢于大胆的选择尝试自己的人生，能有您们这样的父母我真很荣幸，很幸福。爸妈，谢谢这么多年的养育，谢谢您们，您们幸苦了！

最后感谢参与本文审阅的教授、专家们，谢谢你们在繁忙的工作中抽出时间对本文进行批评与指正。